UNIVERSITY OF OSLO
Department of Informatics

The use of a 3D sensor (Kinect™) for robot motion compensation

The applicability in relation to medical applications

Master thesis

Martin Kvalbein

May 2012

The use of a 3D sensor (Kinect™) for robot motion compensation

Martin Kvalbein

May 2012

Department of Informatics
Faculty of Mathematics and Natural Sciences
University of Oslo
Oslo, Norway

The Intervention Centre
Oslo University Hospitable, Rikshospitalet
Faculty of Medicine
University of Oslo
Oslo, Norway

Abstract

The use of robotic systems for remote ultrasound diagnostics has emerged over the last years. This thesis looks into the possibility of integrating the Kinect sensor from Microsoft into a semi-autonomous robotic system for ultrasound diagnostics, with the intention to give the robotic system visual feedback to compensate for patient motion.

In the first part of this thesis, a series of tests have been performed to explore the Kinect's sensor capabilities, with focus on accuracy, precision and frequency response. The results show that the Kinect gives the best measurements at close distance and that the accuracy and precision decreases fast as the distance increases. At $1m$ distance the Kinect could measure the depth between two surfaces with an average accuracy of 96% and standard deviation of 4%. When measuring a moving target there was an average delay of $0.04s$ between actual position and measured position.

Secondly the Kinect was interfaced into the robot control system, where depth measurements from the Kinect have been used to directly control the position of the robot tool. The system could successfully track and follow the motion of a moving surface in front of the Kinect. The system could easily follow motion similar to respiratory motion with a total system delay of less than $0.2s$.

Acknowledgments

I would like to thank my supervisors Ole Jakob Elle and Kyrre Glette for good advice and guidance through this project. Many thanks to Kim Mathiassen for test setup suggestions and good discussion of the results, and thanks to Nguyen Ho Quoc Phuong for helping me understand the robot system. Thanks to Yngve Hafting for providing all the needed equipment and for the help with 3D printing. Thanks to Egil Utheim and Michael Iversen at Mektron, for sharing information about the robot system and the Kinect. I would also like to thank the Intervention Centre at Oslo University Hospital, for providing access to their lab and robotic system.

A special thanks to my wife Martine, you stuck out with me through all my late working hours and got my motivation back when it was gone. And last but not least a big thanks to my lovely son Noah, thanks for being a good sleeper so I could work long hours, and thanks for being enough awake so I could take the best breaks in the world.

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Introduction

Ultrasound diagnostics is a hard and specialized skill. It is a hard physical job as the ultrasound operator has to move the ultrasound probe with high pressure and often work in awkward positions. This may lead to musculoskeletal injuries for the operator [52]. In some cases it also requires high expertise from the operator to get good imaging results[53]. Based on this it is desirable to automate the process by letting a robot do the examination, as this will reduce the fatigue on the operator and allow remote operations if the desired specialized expert is unable to attend the physical site of the examination. Different robotic ultrasound systems are described in [53, 42, 35]. An ongoing Ph.D study at the University of Oslo, *Semi-autonomous robotic system for use in medical diagnostic and treatment*, is working on developing a semi-autonomous robotic ultrasound system that will be used for diagnostics and treatment. The system uses a 6DOF Universal robot and will have haptic feedback and the ability to link preoperative images with intraoperative images. In addition to this, a 3D sensor will be interfaced with the system. This sensor will acquire 3D scene information of the patient's body, and this information will be used to compensate for patient motion to maintain a fixed relation between the patient and the robot. The goal of this master thesis is to see if the Xbox 360 Kinect™developed by Microsoft®is a good enough 3D sensor to be used to monitor the patient movements. To find out this, there are four main objectives that has to be investigated.

1. Sensor accuracy/precision

   How good are the data delivered from the Kinect? Are they accurate and precise enough to be used for robot position control?

2. Sensor bandwidth

   How fast movement can the Kinect track, and how fast can it deliver image information to the robot controller.

3. Robot system bandwidth with senor attached

   How fast can the robot operate when it is controlled with position commands from the Kinect images.

4. Robot tracking capabilities

   How does the Kinect – robot system perform when tracking live motion.

A cooperative partner in the Ph.D study is Mektron[1], a company specialized in robotics for elderly and people in need of care. They have an ongoing project with interfacing a Kinect sensor to a Universal robot, so some of their results will be included in final the discussion of the system.

[1] www.mektron.no

1.1 Project plan

The original description of this project consisted of five main parts that were supposed to be fulfilled in the following order:

1. Literature survey

   3D sensor technology and the use of structured light.

   The use of real time 3D scene information to control a robot.

2. Interface Kinect to Universal robot

   Using existing Ethernet connection for interface to 3D sensor.

   Using low-level robot interface connection to acquire higher bandwidth.

3. Implement software that makes use of the 3D sensor to compensate for the patient movements when examined by a Universal robot holding a probe.

4. Make a demonstration that shows the 3D sensor and the robot working together in the assigned application.

5. Laboratory experiments of the system performance, including acquired accuracy of the 3D information acquired and the performed motion compensation.

At the start of this project the plan went through a set of changes. Since the Kinect is a device ment for gaming, it was found interesting to look into the historical development of game- and robot technology, to see how these two fields of research and technology have developed, and how robot technology now can take advantage of game technology. So the literature survey, background section, mainly focuses on the historical development of robots and video games, before it gives a short introduction to robotics in medicine, 3D technology, robot visual control and the Kinect. Visual control of robots was not prioritized in the literature survey, mainly because the robot system already had implemented a position control algorithm. Since the robot then could be controlled directly by coordinates found in the Kinect data, it was decided to spend more time on experimenting with the Kinect instead.

It turned out to take much more time than expected to get the needed hardware and software to start working with the Kinect. So when all the needed equipment was in place, it was decided to skip directly to laboratory testing of the Kinect's performance, and then continue with the interface between the Kinect and the Universal robot when the Kinect's capabilities had been fully monitored and tested.

When all the tests on the Kinect was completed it was not much time left to implement an interface between the Kinect and the robot. So based on this shortcoming of time, and that the developers at Mektron already had made a setup with the Kinect interfaced to the robot via the Ethernet connection, it was decided to only focus on a low-level interface.

Integrating the Kinect directly into the existing robot control software. From this interface a simple program to make the robot track movements were made. The program was never tested with a patient or the robot blocking the view of the Kinect, but it was used to make the robot tool follow the position of a moving object.

1.2 Thesis outline

The discussions in this thesis follows the same order as the tasks has been carried out in. The two main parts of this thesis has been the investigation of the Kinect's performance, and the interface between the robot and the Kinect.

Chapter 2 gives a background presentation where the development of video games and robot technology is presented, followed by a short introduction to 3D sensor technology, visual robot control and the Kinect.

Then in chapter 3 comes an introduction to the used software. Chapter 4 introduces the tests performed on the Kinect. The tests were designed to give a thorough inspection of the Kinect's sensor capabilities. Test has been performed on a single Kinect and on the combined data from two Kinects. Chapter 5 presents the results from the Kinect sensor tests.

The robot system and the interface between the robot and the Kinect is described in chapter 6, and the results from the systems performance test are found in chapter 7.

In chapter 8 there is a discussion of the overall results and other alternatives that are available along with suggestions to further development. And in chapter 9 the final conclusion is presented.

Chapter 2

Background

2.1 Introduction

The Kinect sensor from Microsoft was intended as a gaming device for the Xbox 360 console. But it did not take long before before robotic researchers discovered the potential in the Kinect. With the Kinect, state of the art depth sensing technology was suddenly available at a low cost.

As a background to this project there will be a quick overview of how the historical development of game and robotic technology. To see what these two technologies has in common, and how robotics now can take advantage of game technology. Then comes a short introduction to the use of robotics in medicine, 3D vision technology and visual robot control. After this, the chapter finishes with an introduction of the Kinect sensor.

2.2 Video and computer Games

To start with, what is a video game? Is there a difference between video games and computer games? There are many discussions about this and an exact answer is hard to give, here the definition will be that all games that use some sort of electronics and a graphical display to work is a video game. The focus will mainly be on game consoles and their development, leaving out hand held game consoles and the old arcade games.

2.2.1 Historical background of video games

There is hard to find any good and reliable sources on the history of computer and video games so this introduction will follow the one given by Wikipedia [2]. This page covers a wide area of the topic and matches what other sources say, so it seems like a reliable source.

The first known electronic game came as early as 1947 when Thomas T. Goldsmith Jr. and Estle Ray Mann constructed something they called a "cathode ray tube amusement device"[3]. The game was a missile simulator where the player's goal was to shoot down airplanes on a display that looked like a world war 2 radar screen. The graphics were poor and the airplanes were manually painted on to the screen.

Four years later in 1951 an inventor, Ralph Baer, came up with the idea of using the television screen as a basis of gaming. But when he passed the idea on to his supervisors in the electronics company he worked for, the idea was rejected.

Later in 1952 a ph.D student at the University of Cambridge, Alexander S. Douglas, made what is believed to be the first digital graphical computer game. It was called "OXO" and was an implementation of the "Tic-tac-toe" game. The game could only be played on one computer at Cambridge so it's popularity was rather limited.

In 1958 a game called "Tennis for two" was designed by Willy Higinbotham. The game was played on an oscilloscope display and simulated a tennis match. This game was the first to allow two players full control of the movement on the screen. The game run on an analogue

computer and calculated the gravity effect on the ball, giving it the correct curves in the ball's path [54, p. 32]. This game was very popular and there were long lines of people who wanted to try the game when the laboratory Higinbotham worked at displayed it. Unfortunately for Higinbotham he didn't see the value in what he had created and didn't take a patent on his invention.

At this time most computers were located at universities and games made were rarely discovered by the public, as most games only were played by their creators, but in 1961 a group of MIT students made a game called "Spacewar!". Here two players tried to shoot each other down with two spacecrafts. "Spacewar!" was distributed over the Internet and is believed to be the first widely available computer game. In 1969 a new version of "Spacewar!" allowed two players to play against each other from different computers, this was the first case of online gaming [4].

In 1966 Ralp Baer could continue his work on his video game idea, now under a secret military project known as "Brown Box" [5]. The idea was that the game could be used in the training of military personnel. This plan failed and in 1972 the concept was sold to Magnavox that gave it the name "Magnavox Odyssey" and distributed it as the worlds first home gaming system. The electronics were analogue and the machine was without any processors or memory. The Oddesey took electronic games out of the arcades and into the home. It was the start of what has today become a multi billion industry. In USA alone, the video game industry made 12.5 billion $ in 2006 [54, p. 1].

2.2.2 Game technology

Game consoles are placed in different generations to separate them based on which time era they are from and what type of technology they use. The following section will use this generational division when to describe how the video game technology has evolved.

The first generation of video games begun in 1972 with the release of the "Magnavox Odyssey". This console had the market for itself for a couple of years, but wasn't very successful. It was not until Atari released a home version of the popular arcade game "PONG" that home gaming took of. The consoles of this generation could mostly play only one game, as the games were "programmed" directly in discrete logic in the consoles. The Odyssey had the ability to change cartridges that would change the electrical circuit and enable different games [2]. There was no real graphics at this time and the games consisted of squares that moved around on the screen, to change the background of the games the players had to place plastic overlays on their TV-screen.

The second generation of video game consoles came in 1976 with the use of ROM (Read only memory) chips and microprocessors. The games were stored on ROM in cartridges that could be plugged into the game

consoles and the microprocessor would execute the games on them [2]. The storage capacity was rather limited, with cartridges that had from 2KB of memory for the first consoles of this generation, and up to 32KB for the last consoles in the second generation. RAM prizes were high and this affected the consoles, that often had less than 1KB of RAM (Random access memory). The CPUs mostly produced 8-bit graphics and ran at speeds from 1 - 4 MHz [6].

Another thing worth a note from this time period is the introduction of the first game featuring a true 3D environment, this was an arcade game called "Battlezone" [54, p. xix].

The third generation of video game consoles emerged in 1984 when Nintendo entered the console market. This generation produced few new features compared to the previous generation, except for more memory and processing power in the consoles, giving better graphics and games. The most important feature of the third generation is probably the introduction of the gamepad as the new game controller, taking over for the old joystick, paddles and keypads. And of course the game Super Mario, featuring Mario as one of the worlds all time most popular game characters [2].

The fourth generation started around 1987 and took us on to 16 bit consoles, which really improved the speed the consoles could operate at. This generation also brought the new CD-ROM technology into the game console market. In 1989 the "TurboGrafx-16" was the first console to come with a CD player. Before the CD, games could be a maximum of 256 KB of code, but now the CD was able to hold up to 550 MB of game code giving the games a great improvement in complexity, detail and sound [54, p. 119]

The fifth generation begun around 1993. In this generation the consoles had 32 bit processors. Some even had 64, like the popular Nintendo 64. More processing power and the new powerful CD technology made 3D games the main focus of this generation.

Sony debuted in the console market with their first console, the Playstation. Sony's console was the first console ever to sell more than a 100 million units [7].

The sixth generation of consoles started with the "Dreamcast", which came in 1998 and was the first console to have a built in modem for online gaming. As always from on generation to the next, computing power increased and the game graphics were improved. In addition to taking online gaming to the consoles, this generation also made the use of alternative controllers popular. Like the guitar in "Guitar Hero" and the Playstation "EyeToy", one of the first successful motion controlled game interfaces [2]. This kind of alternative controllers had been used in previous generations as well, but the technology had been rather inaccurate and they never got really popular.

The seventh generation is the current generation and started in 2005 with the Xbox 360, soon followed by the Playstation 3 and the Nintendo Wii. When Xbox and Playstation focused on HD quality graphics and more computing power in a console than ever before, the Nintendo Wii took a different approach. Aiming at involving a wider range of gamers they had less graphics and smaller games, but their new controller technology gave them a huge portion of the game market. The Wii controller enabled 3D motion detection and gave the player a whole new interaction with the games they played.

After the success of the Wii, Playstation released their "Move" controller in 2010. The two main parts of the "Move" controller is the "Playstation Eye" camera and the handheld wand. The "Move" system can track the player's body movement in three dimensions using the camera and light emitted from the wand, but unlike with the Kinect, the player has to hold the wand in his/hers hand for 3D tracking.

The Xbox Kinect was released in November 2010 by Microsoft to be used with their Xbox 360 console. From its release and until march 2011 it sold over 10 million units, making it the fastest selling consumer electronics device ever[39]. The Kinect features motion control, voice commands and face recognition and is a direct competitor with the Nintendo Wii and Playstation Move. Whats unique about the Kinect is that it allows you to control your video games with just your body, with no need for a hand held controller. Kinect consists of a VGA-camera, depth sensor and a microphone array. It also has a motorized tilt function that allows the camera to find an ideal angle for the best shot of the scene. The technical specifications for the Kinect is further presented in section 2.8

2.3 Robot technology

2.3.1 History

Here follows an introduction of the historical development and the rise of robotic technology. There are a lot of different sources about this topic that all say and focus on different thing, but here we will follow the time line given by [8] and [9]. Any other sources used will be noted in the text.

The idea of automated machines has been living for thousands of years, the first traces of robotic technology can be found as long back as 2000 B.C when Egyptians made automated toys and amusement devices. Later, around 350 to 100 B.C, Greek mathematicians and inventors built automatic birds, door openers and clocks.

In 1350 a mechanical rooster was installed on the top of a church in Strasbourg, it was controlled by a clock mechanism and would flap its wings every day at noon. In 1495 Leonardo DaVinci designed a humanoid robot that looked like an armored knight, the mechanical design was made to look like there was a real man inside the robot.

From the 18th century new developments and machines came quite regularly, most were models of humans or animals that did simple

movements or could play music, write or paint. In 1745 the British inventor Edmund Lee demonstrated the use of feedback when he made a windmill that always was centered against the wind. One interesting "invention" came in 1769 when Baron Wolfgang von Kempelen built an automated chess player, "The Turk", a mechanical man that could play chess. The machine was a good chess player and deceived most people to believe it was real machine with artificial intelligence, when in fact it was operated by a short man in a robot costume. Even though the machine was a fake, it started a discussion of the possibility of machines with artificial intelligence

In 1801, punch cards, which are probably most known as programming devices for the first computers, were introduced by the French inventor Joseph Jacquard. He used the punch cards to control an automated loom.

Later in the 19th century came the introduction of what was to become the foundation of the computer, when Charles Babbage designed and made prototypes of the "Difference Engine" and "The Analytic Engine" in 1822. In 1847 George Boole introduced boolean algebra which today is the basis for the modern digital computer, and thus robot control.

In 1898 came the probably first introduction of a tele-operated robot, when Nikola Tesla demonstrated a remote controlled boat at an electrical exhibition at Madison Square Garden. *"Tesla's device was literally the birth of robotics, though he is seldom recognized for this accomplishment. ... Unfortunately, the invention was so far ahead of its time that those who observed it could not imagine its practical applications."* [10].

The word "Robot" was introduced in 1921 by the Czech writer Karel Capek in his play "Rossums Universal Robots" where human like machines are made to work for the real humans. The word robot comes from the Czech word "robota" and can be translated to "forced labour".

The first master-slave robot system was made in 1951 by Raymond Goertz for the Atomic Energy Commission. The arm was completely mechanical and was used to handle radioactive material and is seen on as an important milestone in force-feedback technology [11].

In 1954 George Devol and Joe Engleberger designed the first programmable robot arm and in 1956 they started working on the worlds first industrial robot, "The Unimate" which took place at the assembly line at General Motors in 1961 where it took die castings from machines and performed welding on auto bodies [37, p.3].

In 1966 work begun on "Shakey" at the Stanford Research Institute. It was one of the first "intelligent" robots and was the first robot that could reason for what it did. Shakey could plan its way from A to B and could move around obstacles. It had no practical use, in the sense that it could not do any kind of work, but it was important for the development of artificial control techniques, robotic vision and path planning [12].

Goertz's master-slave robot, "The Unimate" and "Shakey" are important milestones in robotic development, as they can be seen as the first tele-operated robot, first preprogrammed robot and the first autonomous robot.

2.3.2 Teleoperated Robots

A teleoperated robot, or a telerobot, is a robot that is controlled by an operator from a distance. The robots can differ in how much guidance they need. Some robots need to be told every movement they shall make, while others need only a little guidance to do their work. The latter are a combination of telerobots and autonomous robots. Telerobots are useful when the operator want the robot to do a task in an unstructured environment where it is impossible to program the robot in advance [13].

An important aspect of teleoperated robots is feedback to the operator so that he or she may know what to tell the robot to do next. There are different kind of sensors that can be used to give information back to the operator, but the most important source of feedback is probably vision. There are two main methods to obtain visual feedback. The first and simplest solution is for the operator to have visual contact with the robot so that he can see what the robot is doing at all times. This solution may work under some circumstances, but in many cases the operator don't have the opportunity to be at the same location as the robot. One solution to this problem is to place some sort of camera at the robot and have the robot deliver a live visual feed back to the operator.

2.3.3 Preprogrammed Robots

Preprogrammed robots describes robots that are programmed to do one task, and it does this from you start it and until you turn it of. These are typical industrial robots that do the same repetitive task over and over again. This kind of robot is very useful in a production environment where it can do the same task more precise and faster than a human would have done.

2.3.4 Autonomous Robots

Autonomous robots are robots that can perform a variety of tasks in an unstructured environment without continuous guidance from a human operator [14]. Autonomous robots can be found in many different environments, all from in your garden cutting the lawn or at Mars discovering new territory. Common features with autonomous robots are that they can interact with their environment and make decisions on their own without exterior influence. They do this by taking input from different kind of sensors, like cameras and heat or touch sensors. This input is evaluated by the robot which then acts as it is programmed to do, or the way it has taught itself by machine learning. Autonomous robots can use artificial intelligence to decide and learn what they should do in different situations.

2.4 Game technology in robotics

The main similarity between game technology and robotics is that they both are dependent on computer technology. As computers have become better, so has video games and robots. So it is hard to find any concrete points where one can say *"Video games has done this, and because of that robots are now able to do that"*, as they are both dependent on computer development to move forward. But there is two points I would like to point out where robotic development can take advantage of game technology.

2.4.1 Simulation

Simulation is an important aspect in the development of robotics, especially for mobile autonomous robots. With good simulations more of the development can be done in the simulation environment, lovering cost as the need for hardware and prototypes are reduced. Good simulations is also an important part of video games, as the users want their games to be as realistic as possible. In [21] they describe how to use a simulator based on technology intended for developing computer games to create a simulation environment for their robot.

Video games can also be used to train personnel on how to use a robot. The U.S army uses "The Robot Vehicle Trainer", a video game that simulates combat environments and uses the exact same controls as the real robot they are training to use. This game is used to train soldiers in using a robot to dismantle bombs [15].

2.4.2 Processing power

Game consoles often feature a lot of processing powers, this has the U.S Air Force taken the advantage of. They have built a supercomputer by combining 1760 Playstation 3 consoles. Game consoles are often much cheaper than top modern computer parts and in this case they have used only 5-10% of what a similar system would have cost with off-the-shelf computer parts. The supercomputer is amongst others used for pattern recognition and artificial intelligence research [16], both important things in robotic technology. Of course, the idea of strapping 1000 Playstations to the back of a robot is clearly impossible, but if you could have a wireless link between the robot and the computer you could get a pretty powerful robot.

2.5 Robotics in medicine

In medicine the first robots were introduced around 1985, and were used in neurosurgery [50]. In neurosurgery which requires extreme accuracy, the robot can be used to position the tools and drill the hole trough the skull and into the brain. The robot is less invasive than a human operator and can follow trajectories unreachable to humans. The robots can operate at different levels of autonomy, from only placing the drill used to drilling

the bone, to place the drill, drill the hole and then place the surgical tool. Another early application for robots were in orthopedics where the robot's high precision was used to drill cavities in the bone before a prostheses could be places. The first system for this was the Robodoc [36], which has been used on thousands of patients [50].

These systems were mainly used on solid non-deformable structures that allowed all the movement to be planned in advance. In the 1990s the focus turned to develop more interactive robotic applications, that could be used in situations where it was impossible to plan all the trajectories ahead of the operation. This lead to teleoperation of the robots with the use of master and slave units. There are different methods to control the robot in such systems, one way is the use of a master and a slave unit. In master/slave systems the master unit is controlled by an operator while the slave unit mimics the movements made on the master device. The slave can directly copy the master's movement or adjust the movements by downsampling speed and force, or removing small unintended motions.

The latest trend in medical robotics have been the design of smaller, more light weight robots. These robots can be placed on, or attached to the patient [50]. Smaller robots can easier be integrated in the operating room and are designed to do one specific task. TER [53] is an example of such an on-body robot system. Robots small enough to be inserted into the patient's body is under development. These system will be able to move directly on organ surfaces [50].

2.6 3D imaging technology

There are two general techniques for acquiring 3D scene information, passive and active depth sensing. Passive depth sensing uses the sift in images taken from two different angles to calculate depth. Active methods project some sort of light on the scene, and uses this to measure the distance [28]. Here follows a short introduction to three different depth mapping techniques: triangulation, structured light and time of flight.

2.6.1 Triangulation

Triangulation can be either active or passive [24]. In the active form, a beam of light, normally a laser, is projected onto the scene in sight. Then a camera captures an image of the scene and the position of the light spot is registered in the image. The distance, D, to the object the laser spot was projected on is then

$$D = \frac{B}{\frac{x_0}{f} + tan\alpha} \tag{2.1}$$

Where B is the baseline distance between the laser and camera optical center, x_0 is the position of the laser spot in the image, f is the camera's focal length and α is the angle between the projected laser and the camera's optical axis.

The passive form of triangulation is to use multiple cameras to take images of the scene, this is also known as stere vision. In stereo vision the distance is calculated by the triangulation between the camera positions and matching pixels in the captured images.

Structured light

Structured light is another form of active triangulation. In structured light, a known pattern is projected onto the scene, and the depth is calculated on the basis of the triangulation between a known reference pattern and observed reflected pattern [27].

2.6.2 Time of fligt, TOF

Time of flight is a method to measure depth by using a light source to project light onto the scene. The depth is then calculated from the time delay between a light pulse is emitted and a sensor detects the reflected light. The principle is very simple and the distance D is given by

$$D = \frac{c * \delta t}{2} \tag{2.2}$$

where c is the constant speed of light and δt is the measured time between the light was emitted and the reflection was detected. $c * \delta t$ is divided by 2 since the distance is traveled twice, from emitter to scene and from scene to sensor [26].

Another variant of TOF emits a constant light beam with known period and amplitude. The distance is than calculated by measuring the phase delay of the reflected signal compared to a known reference signal [26].

2.7 Visual robot control

Visual control of robots of robots is a major field within robotic technology. Vision is one of our most important senses and enabling robots to see opens up a lot of potential for the robots.

There are basically two different approaches two visual robot control, position-based visual servo control and image-based visual servo control [47, p. 407]. In position-based control, visual input are used to find 3D position coordinates that can be used to control the robot. The problem with position-based control is to generate these coordinates in real time.

Image-based control uses the image to directly control the robot. The normal method is to define some error function, that based on features in the input image, calculates motion commands to the robot.

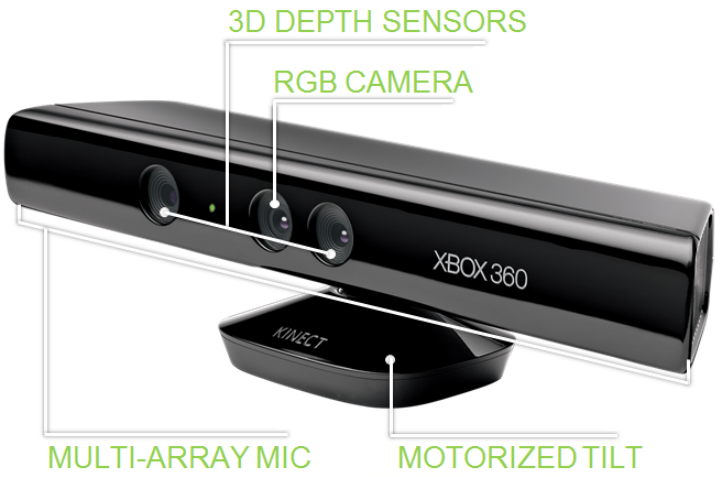


Figure 2.1: The Kinect

2.8 The Kinect

2.8.1 Technical details

The Kinect consists of an infrared, IR, light source, IR light sensor, RGB camera, a microphone array with four microphones, a motor to tilt the sensor and an accelerometer to detect the Kinect's angle relative to the horizon. Both the RGB and IR camera have a resolution of $640x480$ pixels (VGA) and can deliver an image stream of 30 frames per second. In Xbox games the cameras are used to track the players, while the microphone array are used to issue voice commands. Using several microphones the Kinect can locate the source of the sound, and thus it can know which of the players who issued the command. Figure 2.1 shows the Kinect and the location of the different parts. For the depth sensor, the IR-emitter is to the left, while the IR-sensor is on the right. The Kinect uses a class A laser to emit the IR light, and there is therefor no risk of eye injury using the Kinect. The "brain" in the Kinect comes from PrimeSense's *PS1080 System on Chip* [17]. The PS1080 controls the IR emitter and receives input from the cameras and microphones. The depth map are calculated on the chip and all the sensor inputs are synchronized and sent to the host computer via an USB 2.0 interface[38][1]. The Kinect uses more power than the USB connection can deliver, and thus it requires an extra power supply. This power supply is in most case included when you buy the Kinect, since only newer Xbox 360's can deliver enough power to the Kinect.

2.8.2 Resolution

The Kinect has a resolution of $640x480$ pixels on both depth and RGB camera. The field of view for the depth sensor is 57° in horizontal and 43°

[1] This is how the technology works on the PrimeSense sensor, and it is assumed that it is equal for the Kinect

in vertical direction. The infrared light emitted from the Kinect is distorted through a filter, thus giving a "random" placement of the depth pixels. If we assume that the pixel are symmetrically distorted, then the graphs in figure 2.2 on the next page shows how the field of view and resolution changes with the Kinect's distance to objects. The graph in figure 2.2a shows how the field of view increases linearly with the distance. Equation (2.3) shows the calculation of the field of view.

$$Fieldofview = 2 * tan\frac{\theta}{2} * distance \tag{2.3}$$

Equation (2.4) shows how the resolution decreases in x- and y-direction as the distance increases, as shown in 2.2b, here α is 640 for horizontal and 480 for vertical field of view.

$$Pixelspr.cm = \frac{\alpha}{Fieldofview} \tag{2.4}$$

In figure 2.2c we can see that the resolution decreases rapidly as the distance increases, to achieve a theoretical resolution of $1mm$ in the x- and y-direction, measurements would have to be taken at distances below $60cm$. Microsoft reports that the Kinect has a playable range of $1.2m$ – $3.5m$ [34], while PrimeSense reports that their sensor, that uses the same technology can operate at $0.8m$ – $3.5m$ [38]. During tests the Kinect has delivered depth data from $\approx$ $50cm$ and [46] reports that they have measured distances up to $15m$ with the Kinect. So it is clear that the Kinect is operative at a wider set of ranges than reported by the manufacturers.

2.8.3 Depth measuring

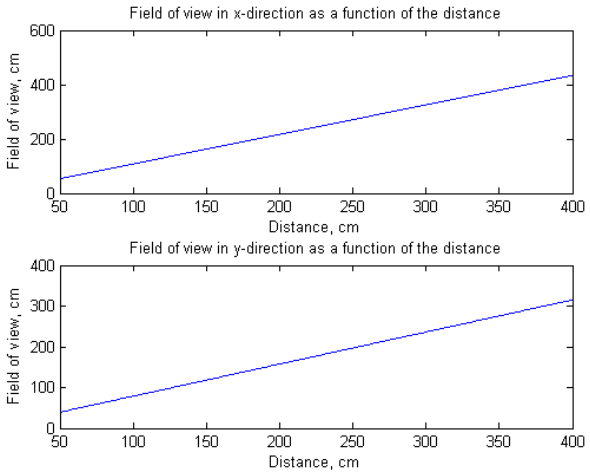
To measure the depth in its field of view, the Kinect uses structured light, or *Light Coding*™ as PrimeSense calls it. IR light with a known pattern is emitted onto the scene, and when the light returns the observed pattern is used to calculate the depth. This is done internally in the Kinect by comparing the observed pattern to a reference pattern stored in the Kinect. Figure 2.3 on page 20 shows how the IR pattern from the Kinect looks like on a flat wall. The reported depth accuracy is $1cm$ at $2m$ distance, and the Kinect uses 11 bits to represents the depth values, limiting the depth resolution to $2^{11} = 2048$ quantization levels.

2.8.4 Programming the Kinect

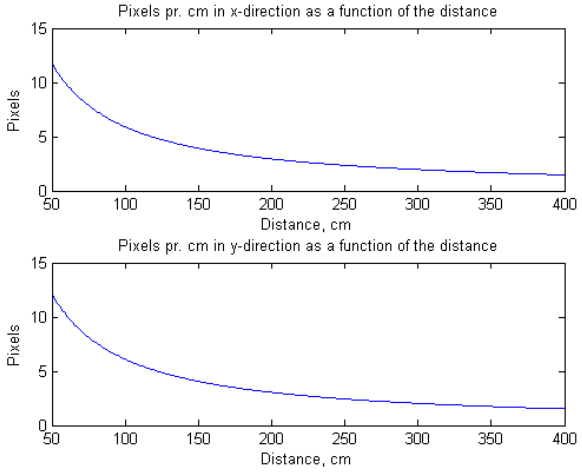
The Kinect was intended as a gaming devise to use with the Xbox 360, but it did not take a lot of time before the first open source drivers for connecting the Kinect to a computer was available online. Many programmers saw the potential in this new technology and wanted to use it. This has lead to open source communities making and sharing code to use with the Kinect.

Today there are three main sources of drivers for the Kinect. The first source of drivers are those made by the open source community OpenKinect[2] and their "libfreenect" software library that provides drivers

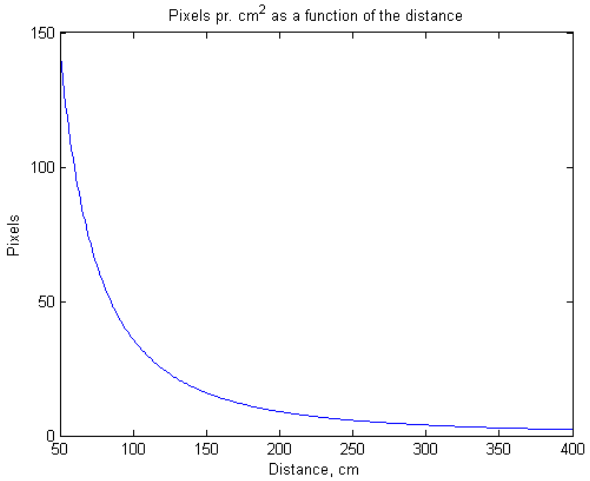
[2] www.openkinect.org



(a) Field of view



(b) Resolution pr. cm



(c) Resolution pr. cm^2

Figure 2.2: Kinect resolution and field of view

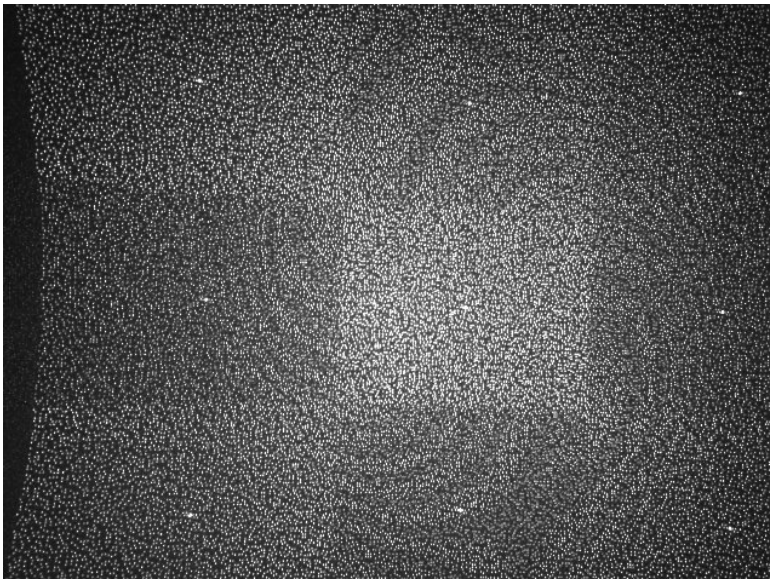


Figure 2.3: Kinect IR pattern, image from [1]

that give you direct access to the raw data from Kinect.

Another alternative software is the software provided by OpenNI, which is an industry-led non-profit organization working with natural interaction between people and machines. One of the main contributors to OpenNI is PrimeSense™, the company that made the 3D technology used in the Kinect.

The last alternative is to use the official drivers released by Microsoft. These drivers were released along with a software development kit, SDK, for the Kinect in May/June 2011, and further updated as Microsoft released their Kinect for Windows in 2012.

2.8.5 In use with Kinect

Computer vision is an important aspect in robotic systems. With the Kinect it is possible to buy an almost complete system for machine vision for just a little money. This opens up for a lot of new research and development as it is now possible to build advanced visual systems without major funding. There has been developed many applications using the Kinect. A great deal of these are not very useful, but some of them really show the range of what is possible to do with the Kinect. Here follows three examples of what the Kinect is being used for today. There are several other applications as well, a good site for looking into them are www.kinecthacks.com.

Robots has been used in the operating room for many years, one major concern with robotic surgery is the lack of physical feedback to the operator. A group of students from the University of Washington has found a possible solution to this [41]. Using the Kinect they have made a program that maps the 3D environment of the robot and defines areas that the robot has to stay out of. When the robot is about to enter an illegal area, the joystick the operator is using to control the robot will stop. This tells the

operator that he is about to do something he should not do. Another great thing about this system is how cheap it is, a similar system without the Kinect would have cost around 50 000$, while you can get a Kinect for around 150$.

Another use of the Kinect in the operating room has nothing to do with robot control, but with image manipulation. In surgery, images helps the surgeon to find his way in a patients body. But if the surgeon need to look at a picture again it might be complicated to explain to another person which picture he wants and exactly where he wants to zoom in. It gets even worse if the surgeon has to go to a computer and manipulate the pictures himself, then he will have to leave the sterile environment at the operating table and then clean up again when he returns. This will take a lot of time in the surgery. A solution to this has now been made, using the Kinect it is now possible to trace the movement of the surgeon's hands in three dimensions. This way he can control the pictures that are being showed at a big screen in the operating room. With just a wave of hands he can change pictures, rotate them and zoom in and out, saving a lot of time for the surgeon [22]. The image control is done just like on the touch screen of a pad or phone, except the motions are done in mid air.

The Kinect has also been used to give visual input to autonomous robots. This is shown in [48, 25], where the Kinect is attached to small quadrotor helicopters, and gives the quadrotors a 3D view of the world. This allows it to fly autonomous around without colliding with other objects.

2.9 Related work

Since its release, it has been done several studies on the Kinect and the quality of its measurements. In [49] the Kinect's depth accuracy is compared to an *Actuated Laser Range Finder, aLRF*, which is a popular depth sensor for mobile robots. In the same test the Kinect is also compared to two other low cost time of flight,TOF, 3D sensors, the SR-4000 and Fotonic B70. Their test indicate that the Kinect gives better readings than the two TOF sensors, and on ranges up to $3.5m$ the Kinect can be a good substitute for the aLRF, as their performance is similar up to this range. In [27] it is shown that the error in the depth measurements increases quadratically with the distance. And they state that for depth mapping applications, the data should be acquired within 1-3 meters distance from the sensor. At larger distances the data quality is corrupted by noise an low resolution. [46] compares the Kinect to stereo vision achieved with two SLR $3.5M$ pixel cameras, and a TOF camera. The Kinect's performance was close to the one delivered by the two SLR cameras, and much better than the TOF. They also modeled the Kinect's depth resolution, $q(z)$, to be a function of the distance z, see equation (2.5).

$$q(z) = 2.73z^2 + 0.74z - 0.58[mm] \tag{2.5}$$

[30] uses a PrimeSense depth camera, built on the same technology as the Kinect to make models of object held by a robotic arm. They show that the camera gives good enough data to recreate 3D models of object surfaces.

Chapter 3

Choice of software

3.1 Choice of Software

As mentioned in the introduction, there are three main sources of drivers for the Kinect, OpenKinect, OpenNI a Kinect for Windows SDK. When the work in this project begun the choice was between the software from OpenKinect and OpenNI, since the Microsoft Kinect SDK had not been released yet. The OpenNI software was chosen because this was the software the programmers at Mektron already had taken into use, and hence it was a natural choice. OpenNI was also the closest alternative to "official" software for Kinect, since Primesense was part of the development of the software. Another advantage with OpenNI over OpenKinect is that OpenNI gives the depth measurements directly in meter units. OpenKinect delivers the raw 11 bit depth values from the Kinect and calibration and conversion to depth values has to be done "manually", while OpenNI deliver an out-of-the-box ready to use solution for the Kinect.

In June 2011 when Microsoft released their SDK little programming had been done in this project so it was a good time to compare the two libraries to see which one was the best to continue with. Both options has their pro's and con's, where the major difference is that the SDK from Microsoft has access to the skeleton tracking algorithm used in the Xbox Kinect games, allowing you to do skeletal tracking without the initial calibration pose. This is a big advantage if you want quick configuration or you do not have the ability to do the calibration pose, which often is the case for medical patients. Table 3.1 on the next page shows a quick overview of some of the main differences between OpenNI and the Kinect SDK from Microsoft[1]. The major advantage with the Kinect SDK was that it supported automatic skeleton tracking, while the advantage with the OpenNI SDK was that it supported different platforms. Based on this it was chosen to continue using the software from OpenNI. This was mainly based on the fact that the robot used in this project is running on a Unix platform, then the software could be installed on the same computer that controls the robot and thus the system can run in real-time. This will remove one source of error and delay as the data don't have to be transferred from one machine to another. In addition to this, there is no need for the automatic body tracking offered by the Kinect SDK, as the focus will be on only parts of the body.

Point Cloud Library

Point Cloud Library [40], PCL, is an open source library for processing 3D point clouds. It contains several algorithms for handling point clouds, such as filtering, feature extraction and point cloud alignment. PCL also has wrappers giving direct access to the OpenNI library, so with PCL you get access to all the same features as with OpenNI.

[1] Note that this comparison was made in June 2011, and hence uses the software available at that time. Both the OpenNI software and the Kinect SDK has later been released with newer versions with more features.

Kinect SDK	OpenNI
Direct support for motor control	No support for motor control
Supports audio commands	No support for audio
Offers full body tracking without calibration pose	Offers body tracking with calibration pose
Only full body tracking	Supports hand tracking and built-in gesture recognition
Only on Windows 7	Platform independent
Can only be used with Kinect	Can be used with Kinect and Asus Xtion Pro
Only non-commercial license	Open source license

Table 3.1: OpenNI vs. Kinect SDK

OpenNI and PCL defines the Kinect coordinate system origin to be at the center of the depth sensor[2]. The z-axis is pointing straight out of the sensor, the x-axis to the left, while the y-axis is pointing straight down.

The OpenNI framework delivers a depth map consisting of $640 * 480$ depth measurements. Each point in the depth map represents the distance from the Kinect depth sensor to the object(s) in sight. The depth values represents the distance from a plane located at the sensor origin, perpendicular to the the depth sensor z axis, and to the objects in sight. In other words, the distance to every point on a flat surface perpendicular to the Kinect's z-axis is the same. OpenNI delivers the depth values in millimeters, while PCL transforms it to meters. PCL uses the depth readings to automatically calculate and make a point cloud, where each point is represented with XYZ coordinates. One can also get the data captured with the depth sensor aligned with the image from the RGB camera. This will give you a RGB-D image, with both color and depth information from the scene in view.

MATLAB®

MATLAB was used to process the data stored from the various tests performed on the Kinect.

Kinect for Windows SDK

There are drives available that can be used to access the Kinect's motor from OpenNI programs. These driver require extra installation and it is not known if you get access to the Kinect's accelerometer with them. Therefor the SDK from Windows was used to control the Kinect's motor. A simple program was made that tilted the Kinect's angle of sight to 0° relative to the horizon.

[2]The sensor is $\approx$ 1cm inside the Kinect. It may seem like the zero coordinate is sifted to the center of the glass plate in front of the sensor.

Solidworks and 3D printer

Solidworks[3] is a 3D modeling software that can be used to make to make 3D drawings that later can be printed out in 3D. In this project Solidworks was used to design three different test models that were printed out on a 3D printer and used to test the Kinect.

[3] www.solidworks.com

Part II

Analysis of the Kinect sensor

Chapter 4

Test methods and setup

In robotics and especially in medical applications it is vital that the equipment used is reliable. It is important to get high accuracy and precision. A set of tests was done to find out more about the Kinect's accuracy and precision. This chapter gives an introduction to all the tests performed on the Kinect. The results are presented in chapter 5 in the same order as they are given here, so test will be referenced by their section number.

There have been made many reports on the Kinect's accuracy and precision, but they have all tested the Kinect on objects that were standing still. In addition to test the Kinect on still standing objects, a method to test the Kinect's tracking capability of a moving object is presented in this chapter.

A series of test were performed, which most of followed the same structure:

1. Initialize Kinect
2. Wait for depth measurement drop
3. Buffer N frames
4. Filter out region of interest from the buffered frames
5. Save data to file
6. Export data to MATLAB
7. Process data in MATLAB

Step 2 in the test structure comes from the fact that after the Kinect has started capturing frames, it uses some time to calibrate before it finds the correct depth. This can be seen in figure A.1 in appendix A on page 105. There seems to be a change in the initial measured depth after around 40 frames which is approximately 1.33s. So this measurements change was avoided by manually starting the buffering in each test after a short period of time. In all of the tests the Kinect was positioned against a flat wall or some test object, like shown in figure 4.1 on the next page.

The first tests are intended to find out more about the Kinect's technical details, if the measured resolution corresponds with the reported specifications. Then there are some tests to investigate the Kinect's precision and accuracy. After that, data from the Kinect is used to recreate a sphere, to find out if the Kinect data can be used for object estimation. Then an estimation of the angle a surface must have relative to the Kinect to be seen by the depth sensor is performed, followed by a test on the Kinect's capabilities to measure the position of a moving object. Finally a method for aligning data from multiple Kinects are presented and tested.

Since the Kinect is most accurate on close distances [27], and the depth resolution has an exponential increase with the distance [46], most of these test has been taken at the range of $0.5m$ – $1.5m$. Some tests, when the available space has allowed it, has been done with ranges up to $4m$.

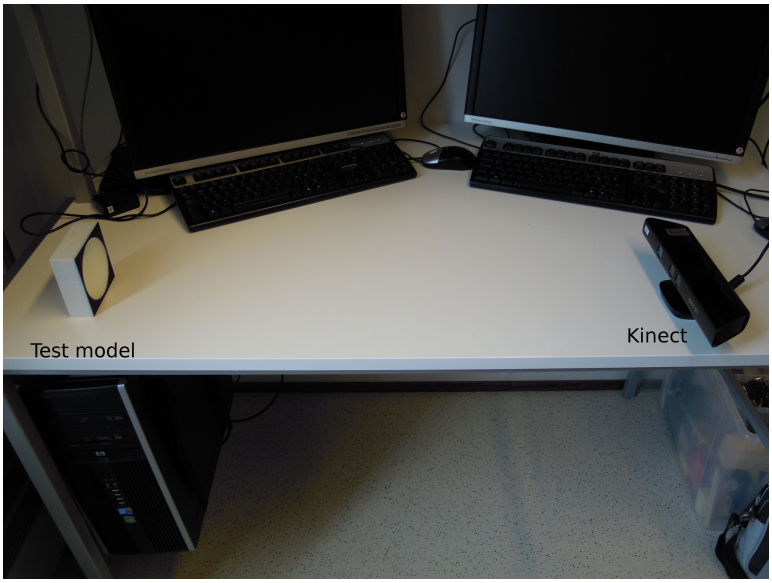


Figure 4.1: Typical Kinect – test-model setup

4.1 Resolution

4.1.1 Depth resolution

As mentioned earlier, the Kinect's depth resolution seems to decrease as the distance increases. So to confirm [46]'s model of the depth resolution, a similar test was performed. The Kinect was placed in front of a flat wall and as the Kinect was slowly moved away from the wall, all the depth measurements from the Kinect's center pixel were stored. Then the data were transfered to MATLAB where all the reported depth values were sorted and the step size between each depth value were calculated. The step size would then indicate the Kinect's depth step size. To make sure that the Kinect was moved slow enough to not skip any depth values, the test was done four times and all the data from the four tests were combined.

Improving the depth resolution

Instead of reading the depth from just one single pixel, one can use the average of several pixels to give an estimate of the depth. This will increase the depth resolution. One way to do this is to define a grid in the (x,y) space and then average all the depth measurements within this grid into a new single depth value. This will also decrease the number of data points, which may be a good thing if you are short on processing power. Of course, this comes at the price of lower resolution in the (x,y) plane. Another alternative, if you don't want to loose resolution, is to use an average filter on all the collected data. Then the number of points will be the same, but their depth values will be made from the average depth of their neighborhood. This solution will need more processing power, but will keep the point density. The downside of this approach is that depth

edges will be smoothed out.

Based on this method a second resolution test was performed, with the same setup as before, but this time the depth step size were calculated from the average depth from all the pixels within a $1cm^2$ square at the center of the Kinect's view.

4.1.2 XY resolution

In section 2.8.2 we saw how the Kinect's resolution in the x- and y-direction decreased as the distance between the Kinect and objects increased. This was a theoretical resolution calculated from the Kinect's angular view and reported resolution. This test is meant to control these calculations. The Kinect was placed in front of a flat wall and a $20x20cm$ square at its center of view was filtered out. Then the number of depth measurements made from this square was counted up. This was done for a 100 frames and then the Kinect was moved further away from the wall and the same procedure was repeated. The average number of data points captured in these 100 frames at each distance were used to compare the measured resolution to the estimated resolution.

4.2 Stability

4.2.1 Time stability

One of the great advantages in using a robotic system, is its capability to deliver the same results over a long period of time. Therefor it is important that the measurements from the Kinect is stable over longer time periods. To find out how the Kinect performs over time, it was placed against a flat white wall. Then a $20x20cm$ square in its center of view was filtered out and the depth measurements from this square was stored for every 30'th frame, i.e. one frame pr. second. This was done for three different runs that lasted 10, 20, and 60 minutes. The captured data was transfered to MATLAB, and the average depth to the square was calculated for each frame. Since the Kinect stood still in front of the wall during the tests, all the depth measurements should be more or less equal. The results from this test is presented in section 5.2.1

4.2.2 Image stability

The Kinect it sold as a cheap consumer electronic, so the quality in the electronics may be variable. For this project it is important that the depth measurements stay stable and reliable. This test was performed to find out more about how the quality of the depth images is over the entire field of view, if any part of the image is better than others. This was done by placing the Kinect directly in front of a flat white wall, the Kinect was placed close enough to the wall so that nothing else than the wall was in sight. Then all the captured depth readings from the Kinect were stored for a 100 frames. In MATLAB this data was used to generate two images, one depth

image containing the average depth to the wall for all the 100 frames, and one image made by counting up the number valid depth measurements for each pixel in all the 100 frames. The depth image was used to see if the average depth to the wall was different in various regions of the image. Since the wall is vertically flat and perpendicular to the Kinect's line of sight[1], the depth should be the same for the entire picture. To get an indication of how the measurements changed in the depth image, the average depth and standard deviation along a horizontal and vertical line starting from the image's center and to the top and right edge of the image were calculated. The image showing the valid number of measurements can be used to see where in the Kinect's field of view it looses most data, if any. Based on the results in the "missing data" image, another 100 frames of data were captured, with the Kinect looking out over a random office scene.

4.3 Measurement precision

The meaning of this test was to find out more about the Kinect's precision at different lengths from a surface. Precision is the ability to produce the same results over and over again. By calculating the average depth from the measurements, the standard deviation can be used to estimate the precision. The Kinect was faced against a flat wall, and the depth measurements returned from the sensor was filtered so that only a $20x20cm$ square, with the center at x- and y-coordinates $(0, 0)$ was used in the further calculation. The depth readings from all the pixels in this square for 100 frames was saved for further processing in MATLAB. For each 100 frames, the sensor was moved a bit further from the wall. A total of 7 tests were performed, on a distance from $56cm$ to $200cm$. In MATLAB the average distance and standard deviation to the wall was calculated from all the depth readings in a frame. A similar test was done in [27] and they found the error to increase quadratically with the distance.

4.4 Accuracy

Accuracy is the ability to make measurements that are close to the real values. Here follows two tests intended to find out more about the Kinect's measuring accuracy in the depth space, and in the 2D xy-space.

Since the resolution decreases when the distance increases it is expected that the accuracy will decrease with the distance as well. But since the reported results are the average measured lengths from 100 frames, there is a possibility that the accuracy will stay approximately the same, while it is the precision that has the largest decrease. This is because it is assumed to be equally probable that the measured distance is to long as it is to short.

[1]The Kinect's built in accelerometer and motor was used to align the Kinect's z-axis to a horizontal plane.

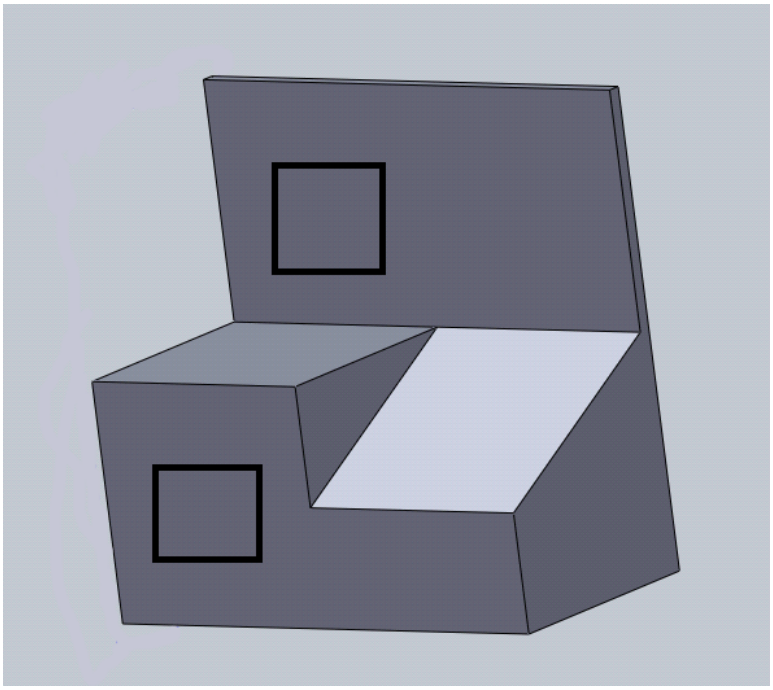


Figure 4.2: Test model used for depth and edge accuracy testing

And over a 100 frames this may even out to give the correct lengths in average.

4.4.1 Depth accuracy

The most important feature in the Kinect is it's depth measurements. So therefor it is important to know more about the absolute depth accuracy. One obvious way to measure this would be to place the Kinect in front of a surface, measure the depth with the Kinect and then do another depth measurement with another device and then compare the results. This is what is done in [49], where they compare the Kinect with an *Actuated Laser Range Finder, aLRF*. In this project, the only other measuring device available was a carpenter's ruler, and attempts to measure the same distances with this turned out to be a major source of error. As a solution to this, a test model (figure 4.2) was designed in Solidworks and printed out in 3D. The distance between the closest surface and the furthest surface on this model is exactly $5cm$. The initial plan was to let the test program filter out a $2x2cm$ area on both of the surfaces, and then the average distance to the pixels within these areas could be calculated. But as the distance between the Kinect and the model increased it was hard to get good enough manual alignment of the Kinect and the model, so these measurements became to inaccurate to be useful. Instead the Kinect took depth readings from the entire figure and the data was passed on to MATLAB where the regions of interest were manually located and filtered out. These regions are illustrated by the black squares in figure 4.2. 100 frames of depth data from the Kinect was captured and for each frame the distance Δ_z was

calculated with equation (4.1), where z_1 and z_2 are the average distance to all the pixels within the furthest and closest square.

$$\Delta_z = z_1 - z_2 \tag{4.1}$$

The measured depth difference, Δ_z, was calculated for each of the 100 frames at each distance between the sensor and the model.

4.4.2 XY accuracy

This test was made to find out more about how well the Kinect could measure distances in the x- and y-directions. When the depth images from the Kinect has been visually inspected, it has been clear that there is a lot of noise around sharp edges.

The test was performed by placing a A4 paper on a window[2] and then place the Kinect right in front of the paper, making sure that the center point in the sensor depth image was located on the paper. Starting at the center pixel a horizontal and vertical line was drawn until they reached the end of the paper. The distances between these end points were used to calculate the width and height of the A4 paper[3]. The found distances was saved, along with the measured distance to the center point. As before, this was done for a 100 frames and the distance between the Kinect and the paper was increased for each test run, ranging from $0.5m$ – $4m$.

4.5 Edge accuracy

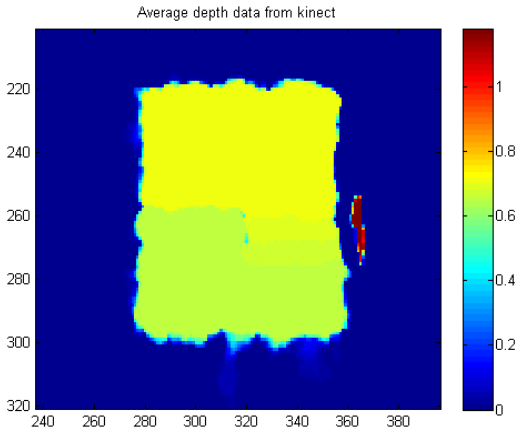
For this test the same model as in the depth accuracy test was used. If you look directly at the front of this model you will see one sharp edge and a tilted surface with two less sharp edges. These edges can be used to compare how the Kinect performs around edges and surfaces with different angles. The model was placed directly in front of the Kinect, as perpendicular to the Kinect's z-axis as manually possible. Then a hundred consecutive frames captured from the Kinect was buffered, and the $20x20$ cm area in the center of view, which contained the model, was filtered out and the depth data was stored for further processing in MATLAB.

In MATLAB the stored data was used to make a depth image as shown in figure 4.3a on the next page. The image is made of the average depth in all the 100 frames. On this image the sharp and tilted edges were manually located and filtered out as shown in figure 4.3b and 4.3c, these regions were selected with a margin to avoid disturbance from the vertical edges.

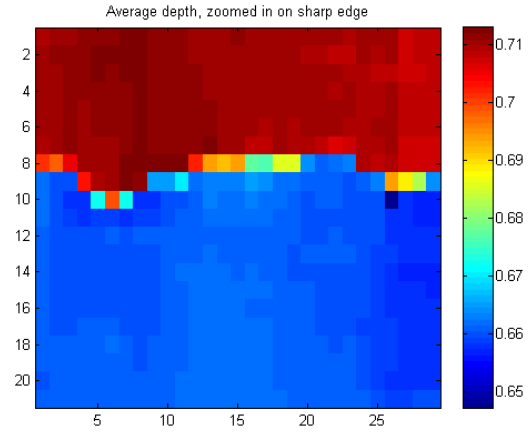
The edges around the model and the sharp edge in the middle of it should have been straight lines and edges, but as we can clearly see, this is not so. In figure 4.3b we can see the depth difference between the closest and furthest surface as the red and blue areas. Ideally the transition

[2] A window was selected since this would make it easier to segment the object from the background, since the Kinect does not detect the glass surface.

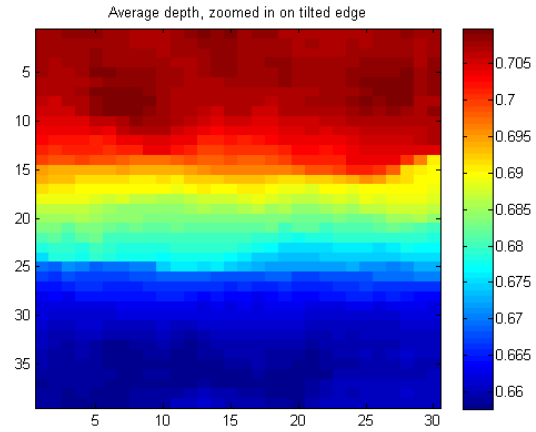
[3] The paper was positioned in a "landscape" orientation, so the width is the longest part of the paper



(a) Depth image of the test model



(b) Depth image zoomed in on the sharp edge



(c) Depth image zoomed in on tilted edge/surface

Figure 4.3: Depth image of the test model used to test edge accuracy. The numbers on the axis' are just pixel indexes added by MATLAB

between the red and blue area should be a straight line, but here there are three rows of pixels that indicates the edge. This test was done for 8 runs where the distance between the Kinect and the model was increased with around 10cm pr. run. To say more about the edge accuracy, the average depth at pixel row was calculated. The standard deviation at each row can the indicate how well the Kinect detects edges.

4.6 Model reconstruction

One alternative to use single data points from the Kinect to control the robot is to let the images from the Kinect be used to make a model of the object is sight. Then this model can be used to control the robot in some sort of way. This test is intended to find out how good we can recreate a model with data captured from the Kinect. A similar test has been done in [30], but they have used a PrimeSense depth camera instead of the Kinect. Their test showed good results so it was of interest to see if the Kinect could do the same.

This test was done on two different models. The first model was a full sphere, and the second model was a half sphere. With the full sphere the shape could be measured from a convex surface, while with the half sphere the data were captured on the concave side of the sphere. The Solidworks drawings of the shapes can be seen in appendix C.

4.6.1 Convex sphere

A sphere with outer radius of 5cm was designed in Solidworks and printed out in 3D. The choice of designing and printing a sphere instead of just using a simple ball was taken to ensure that we knew exactly the size of the sphere. This sphere was placed directly in front of the Kinect, the positioning was made so that the sphere was the only visible object to the Kinect within a certain range. This made it easy to filter out the background and capture data only from the sphere. For every data point captured from the sphere, the x-, y- and z-coordinates were stored. For each run a 100 frames of data were captured, and the test was done at 10 different distances between the Kinect and the sphere. In MATLAB the saved data were used to make an estimation of the sphere radius and the sphere's position in the Kinect's coordinate system. This estimation was done with a variant of the least squares method, found at [18].

In addition to estimating a sphere model, the captured data was used to make an estimate of the angle a surface must have relative to the Kinect to be visible for the depth readings. This is further explained in section 4.7.

4.6.2 Concave sphere

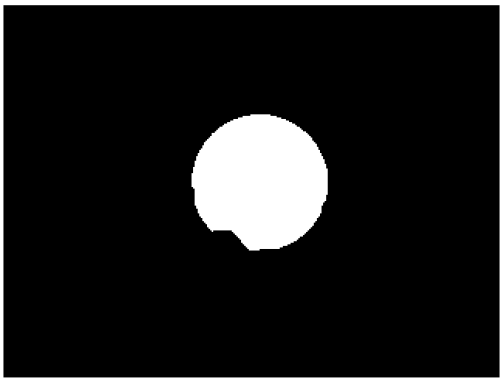
This time a model representing the concave side, or the inside, of a sphere was used in the measurements. The model curvature represents 60% of the inside of half a sphere with an inner radius of 5cm. In the test with the full

sphere it was easy to place it so that everything else could be filtered out. With this half sphere model this was not possible, mainly because of the flat frame around the sphere. To overcome this problem, the flat edge was colored black, while the region of interest, the half sphere, remained white. Since OpenNI/PCL has a built in functionality to register the RGB image with the depth image, the RGB image can be used to locate the sphere and segment the correct data from the depth measurements. Figure 4.4a on the facing page shows the on of the RGBD images captured by the Kinect. Using MATLAB, the RGB image of the half sphere was converted to a binary image. Since the image only consisted of the $20x20cm$ square in the Kinect's center of view, the largest object in sight was the sphere, and hence every small object (background objects behind the model and noise) could easily be removed from the image. The Kinect's depth data often consists of a lot of noise and thus leaving some holes in the sphere surface, these holes were easily filled with MATLAB's *imclose* method. The image in 4.4b shows the circle segmented from the RGB image. It has been observed a slight displacement in the alignment of the RGB and depth image, so to avoid that some of the background is included in the found circle, it is slightly shrunken to guarantee that the background will not be included in the sphere data. Now the sphere can be filtered out from the depth data, using the circle found in the RGB image as a mask, figure 4.4c shows the remaining depth data after segmentation and masking. As in the previous test, the sphere center and the radius was calculated using a variant of least square method. To take the measurements, the model of the half sphere was placed in front of the Kinect, with the Kinect's center of view approximately at the center of the sphere. Then all the measurements outside a $20x20cm$ square centered at the center of view were filtered out. For each setup, RGB and depth data were stored for a 100 frames. Then the model was moved approximately $10cm$ further away from the Kinect for each take. Image 4.1 at the start of this chapter shows this test and setup.

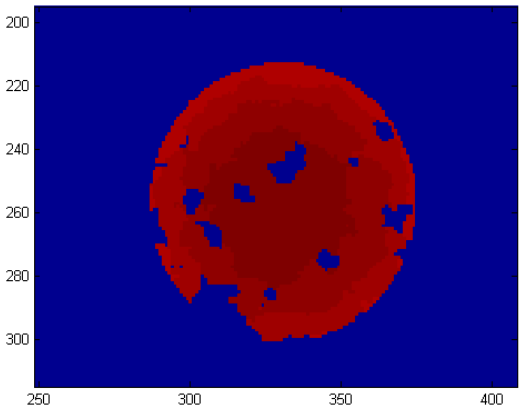
4.6.3 Comparison of convex and concave surface

In the next chapter the results from these two tests are presented. There we can see that the results are better for the data captured from the concave side of the sphere, compared to the data from the convex side. But as stated by [27], and later shown in the test described in 4.9.1, the environment around the object may have an impact on the measured data. These two tests were made with two different setups, the full sphere had no other objects close to it, while the half sphere stood on a flat white table. So these environmental differences may affect the results and thus corrupt a comparison between the two results. So to get a good comparison a third test run was done. This time the full sphere was placed inside the half sphere, so that the sphere could be segmented from the image the same way as the half sphere.

(a) RGBD image from the Kinect



(b) Segmented half sphere from RGBD image



(c) Depth image filtered out with the segmented sphere

Figure 4.4: Steps in segmenting a sphere from the Kinect's RGBD image

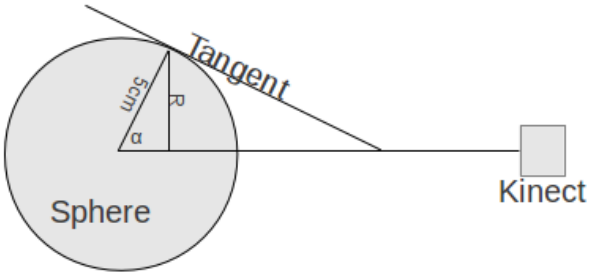


Figure 4.5: Sketch of how much of the sphere the Kinect can read depth data from, and the angle this corresponds to

4.7 Angle of sight

Based on the data from the test in section 4.6.1, an estimate of the angle the surface of an object must have relative to the Kinect to be visible to the Kinect, can be established. The sphere has a radius of $5cm$, so by looking at the sphere in a 2D view one would see a circle with $5cm$ radius. In figure 2.2c on page 19 we saw how the number pixels pr. cm^2 decreased as the Kinect was moved further away from the object. By counting the number of pixels that the Kinect could read from the sphere we can estimate the radius of the circle this measurements fits with. From this radius we can make an approximation of the maximum angle an object can have relative to the Kinect. Figure 4.5 shows a simple sketch of the calculation, where R is the estimated radius. We are interested in finding α which is the angle between the Kinect's z-axis and the normal of the tangent of the points on the sphere were the circle with found radius is. α is given by

$$\alpha = sin^{-1}(\frac{R}{5}) \tag{4.2}$$

When α is found, it can be defined that the angle between the Kinect's z-axis and the surface normals of the object in sight has to stay within an angle of $\pm\alpha$ for the Kinect to be able to take depth measurements from the surface.

The data for this calculation are taken from the first test with the full sphere. In this test the sphere was the only visible object to the Kinect, so all the captured data points can be used. In the test where the sphere was segmented out based on the background color, some data points may have been lost, and thus the estimation of the radius would be wrong.

4.8 Kinect's frequency response and bandwidth

A system's frequency response can tell us about the stationary relations in the system, while the bandwidth tells us about the operative range of the system. In [51, p.90-97] a system and its response, $y(t)$, to an input signal is given as shown in equation (4.3). The input is a sine wave with amplitude u_0 and frequency ω. $H(j\omega)$ is the frequency response and $\angle H(j\omega))$ is the system's delay.

$$y(t) = u_0|H(j\omega)| * sin(\omega t + \angle H(j\omega)) \tag{4.3}$$

The numerical value of the frequency response (4.4) is the ratio between the output amplitude y_0 and the input amplitude u_0. In chapter 7, where the robot system's bandwidth is tested, u_0, y_0 and $\angle H(j\omega))$ are shown in figure 7.1a on page 81.

$$|H(j\omega)| = \frac{y_0}{u_0} \tag{4.4}$$

From [51, p.97] we have a definition of a system's bandwidth: *The frequency response decreases as the frequency on the input signal increases. Finally we reach an input frequency that doesn't give a response. This kind of system is called a low-pass filter, and the frequency specter where the system responds to the input signal is called the system's bandwidth.* From [51] we also get an algorithm to test a system's frequency response, and find the bandwidth. This algorithm is shown in alg. 1.

Algorithm 1 Algorithm to find a system's frequency response

1. Input a sine wave with frequency ω_i and amplitude u_0
2. Wait until the output is stable
3. Measure output level y_0
4. $|H(j\omega_i)| = \frac{y_0}{u_0}$
5. Repeat 1 - 4 for ω_0 to ω_n

4.8.1 Kinect's response and bandwidth

The Kinect operates at 30Hz, so according the the sampling theorem, that states that sampling has to be done with a frequency at least twice as high as the original signal frequency, an assumption can be made that the Kinect can track movement at up to 15Hz. Figure 4.6 on the next page shows a theoretical estimation of the Kinect's response to movement with increasing frequencies, from $1Hz$ – $15Hz$. The estimation is done by taking samples at 30Hz from a sine wave of increasing frequency, as described in algorithm 1. Normally the response is shown in a bode plot where the magnitude of the output is visualized along with the phase delay of the output signal. But here the samples are taken directly from the sine, and therefor it is no delay. So for simplicity the response is plotted in a "normal" plot where the x-axis is the input frequency and the y-axis is

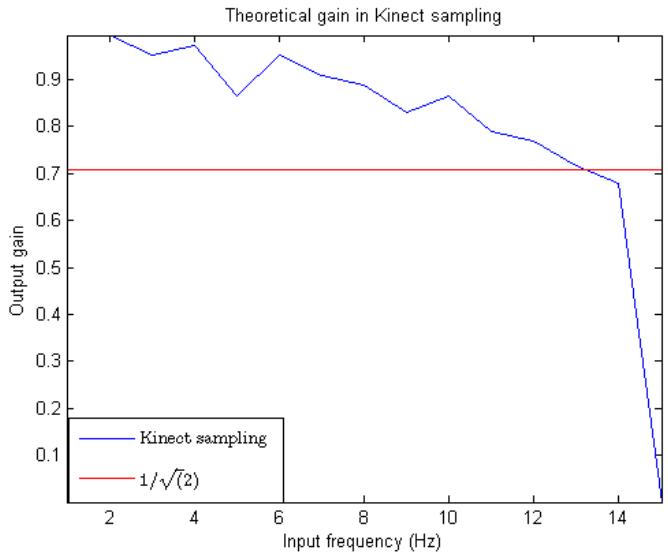


Figure 4.6: Theoretical gain from sampling a periodic sine with the Kinect

the output gain, see equation (4.4). The cut-off frequency in a system is normally defined as where the output response has been damped with a factor of $-3dB$ relative to the input signal. $-3dB$ is the same as a $1/\sqrt{2}$, so $1/\sqrt{2}$ is marked in the plot. As the input frequency increased, the sampled output signal became quasi-periodic for some frequencies, in these cases the output amplitude is defined as the average height of the peaks in the sampled signal.

To test the Kinect's actual bandwidth, the robot system which is described in chapter 6 was used. The Kinect was placed directly under the robot's tool, and the tool was controlled with a periodic sine wave so that it moved up and down along the robot's z-axis. The robot tool position was measured with the center pixel in the Kinect's depth image. For every new frame received from the Kinect, the measured robot tool position was stored along with the last actual tool position reported by the robot. The Kinect operates at $30Hz$ while the robot position is updated with $125Hz$, so it is a theoretical delay between the Kinect's data and the robot's data of $1/125s$. For the Kinect to be able to see the robot tool, the tool has to be at least $50cm$ away from the Kinect, so the Kinect has to be placed below 0 according to the robot's z-axis. To calibrate the measured position from the Kinect with the real position of the robot tool the Kinect ran for a few seconds, capturing depth data while the robot tool stood still in the default start position (z-position 0.45). The data captured from the unmoving robot tool, was used to calculate a bias that was subtracted from the Kinect's measurements to get the real robot tool position.

$$Bias = MeasuredDefaultPosition - 0.45 \tag{4.5}$$

$$RobotPosition = MeasuredRunningPosition - Bias \tag{4.6}$$

At the starting position the robot's tool was approximately $74cm$ away from

the Kinect. We saw in section 5.1 that at this distance the Kinect's resolution varies between $1mm$ and $2mm$ so it is clear that there will be some errors in the measured positions.

Ideally the robot should have moved with increasing frequencies from $0Hz$ to $15Hz$ to test the entire range of the Kinect, but as we will see in the robot bandwidth test in chapter 7, the robot couldn't move any faster than $3Hz$ so the test only goes from $0.25Hz$ to $3Hz$. During the test also the amplitude of the robot movement was increased, but with increasing frequencies the amplitude had to be decreased for the robot to be able to produce a stable motion. More about this behavior is described in chapter 7.

4.9 Multiple Kinects

On way to increase the system's field of view, and hopefully increase the performance, is to use multiple Kinects to capture data. This section focuses on how the data from two Kinects can be combined, and looks into the effect of having two Kinects projecting IR light over the same surface.

4.9.1 Interference

One major concern with using more than one Kinect is that they may interfere with each other. As mentioned before, the Kinect emits its own IR light to measure the depth, but what happens if there is more than one Kinect casting light over the same surface? Will the Kinects disturb each other, or will they be able to separate their own light pattern from the other.

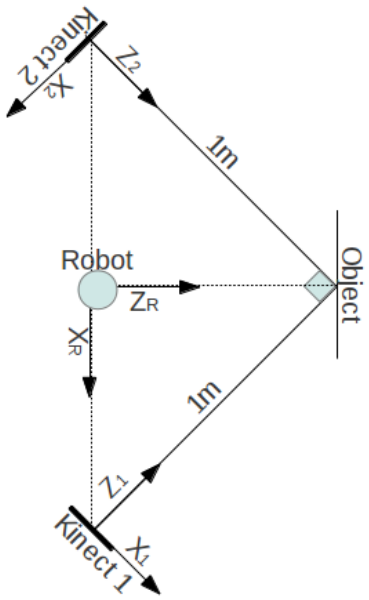
In this setup one Kinect was placed on top of another, and in front of the Kinects a box was placed with a $15.5cm * 34cm$ visual surface. With this placement, 100 consecutive frames of depth readings from each Kinect was buffered, and for each frame the number of pixels containing valid depth measurements was counted up, and the average and standard deviation from these $2x100$ frames were stored. This was done three times with three different configurations:

- Both Kinects emitting IR light
- IR emitter on the lower Kinect blocked
- IR emitter on the upper Kinect blocked

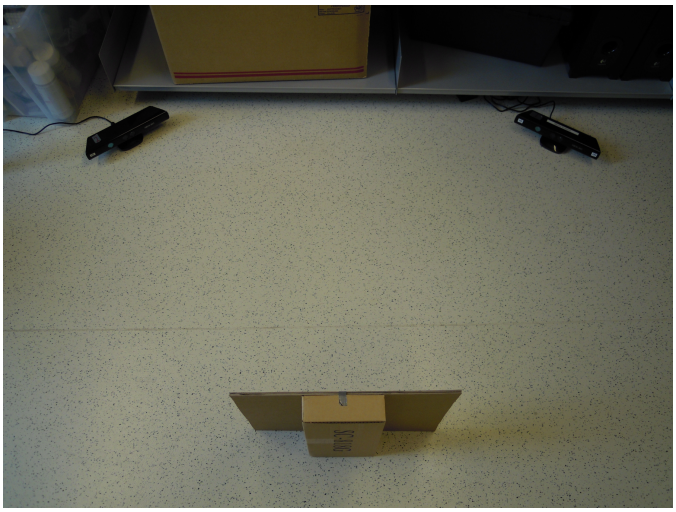
In all three cases both the Kinects' IR sensors were active the whole time.

It was observed that it could seem like the surface that the object in sight was standing on had an impact on the results. Therefor the tests was first performed with the box alone on the white floor, and then with a cardboard plate lying in front of the box on the floor. This was to see if the cardboard plate would change the reflection from the floor and affect the results.

To see if the angle between the Kinect's had anything to say on the interference between them, the same six test runs were run again with a different placement of the Kinects. This time the Kinects were placed with



(a)



(b)

Figure 4.7: Example setup for testing with two Kinects

an angle of 90° between their z-axis, looking at the same target at a distance of $1m$, see figure 4.7 on the facing page. Figure 4.7a also shows how a robot can be placed between the two Kinects. The image in figure 4.7b shows the setup for the test described in section 4.9.4, for this test it was only the standing box that was visible to the Kinects, the cardboard plate was either lying on the floor, or completely removed.

4.9.2 Image registration

Registration is the process of aligning images of the same scene taken from different angles. With registration all the images can be combined to one large image of the entire scene. There are several different ways to align depth images. If you know the positions where the images were captured from, you can use this to calculate the needed transformation. When the images have overlapping areas, one can select a set of key-points in every image and find the transformation with them. Another approach, if you do not know the position of the cameras, or you don't have the possibility to manually select key-points is to use some sort of automatic registration [33]. In this situation, it is assumed that the Kinects will be placed at a fixed location, allowing precalculation of the transformation needed to align the data.

One possible setup with the Kinect's and a robot is shown in 4.7a on the preceding page. The x- and z-axis is shown for all the respective coordinate systems. In all cases the y-axis is pointing down, making a true right-hand coordinate system for each. With this setup the Kinects' coordinate systems have to be transformed to match the robot coordinate system. To do this we have to rotate and translate the coordinate systems. Rotation has to be done around the y-axis, and can be done with the rotation matrix shown in (4.7)

$$R_{y,\theta} = \begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix} \tag{4.7}$$

For the first Kinect $\theta_1 = -90°$ and for the second Kinect $\theta_2 = 90°$. After rotating the coordinate systems they have to be translated. That can be done with the translation matrix show in (4.8), where δx, δy and δz is the distance we want to move the coordinate systems in the x-, y- and z-directions.

$$T = \begin{bmatrix} 1 & 0 & 0 & \delta x \\ 0 & 1 & 0 & \delta y \\ 0 & 0 & 1 & \delta z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.8}$$

We can combine (4.7) and (4.8) to get the final transformation matrix (4.9), allowing the rotation and translation to be done in one calculation.

$$R_{y,\theta} = \begin{bmatrix} cos\theta & 0 & sin\theta & \delta x \\ 0 & 1 & 0 & \delta y \\ -sin\theta & 0 & cos\theta & \delta z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.9}$$

Applying this to the setup shown in figure 4.7a on page 44 we get (4.10) for the first Kinect and (4.11) for the second Kinect.

$$R_{y_1,-90} = \begin{bmatrix} 0 & 0 & -1 & -\frac{\sqrt{2}}{2} \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.10}$$

$$R_{y_2,90} = \begin{bmatrix} 0 & 0 & 1 & \frac{\sqrt{2}}{2} \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.11}$$

In an ideal world this would be enough to align the two point clouds from the two Kinects, but unfortunately it isn't that simple. There are other sources of error that have to be taken into consideration. For instance, the Kinects may be slightly misplaced, causing a different angle than 90° between them. This will lead to misleading depth values, and erroneous control of the robot. To adjust for this error another extra method has to be used to achieve fine alignment of the point clouds. So after the initial transformation that hopefully brings us close to the correct alignment, we can use the *Iterative Closest Point*, *ICP* [19] algorithm to achieve the final alignment of the clouds.

Finding the correct transformation matrix to align the data stream from the two Kinects uses a lot of computation power and is not always applicable to do in a real-time data stream. Therefor a transformation matrix has to be found before the system is to be used, and then this transformation has to be used on the data streams in real-time. Since the Kinects IR signals may disturb each other, it will also be a good idea to keep the Kinects at fixed positions so they can be run one at a time to avoid the interference during registration. In the next subsection a proposed algorithm to align the data from two Kinects with fixed positions is presented.

4.9.3 Suggested algorithm for aligning depth images from two Kinects

This algorithm (alg. 2 on the next page) assumes that the Kinects are placed at fixed locations and will not be moved during or after the alignment process. One of the Kinects has to be defined as the master unit, and after the manual alignment the other Kinect has to be fine aligned to this through the ICP algorithm. *transmat1* and *transMat2* are the transformation matrices calculated from Kinects' fixed positions relative the desired new origin. If the wanted new origin is the same as the origin of one of the Kinects then one of the matrices can be left out or be defined to the unity matrix. To improve the results of the ICP algorithm, it may be a good idea to filter out some of the noise in the Kinect measurements. Further downsampling of the data can be used to reduce the computation time of the ICP. The *bestScore* variable is used to determine which of the transformation matrices

found by ICP is the best. PCL's implementation of ICP gives a score to the transformation based on the euclidean distance between the points in the two clouds, so the lower score the better. *finalTrans* is the transformation matrix found by ICP. So to align the data in the real-time stream, the data from the master Kinect must be transformed with the predefined transformation matrix, and the other Kinect with the matrix found by ICP.

Note that this procedure only aligns the data from the two Kinects, further calibration has to be done to align the Kinect data with the robot's coordinate system.

Algorithm 2 Algorithm for aligning data stream from two Kinects

```
maxBufferSize ← n
Start Kinect 1
while bufferSize1 < maxBufferSize do
    Fill buffer 1
end while
Stop Kinect 1
Start Kinect 2
while bufferSize2 < maxBufferSize do
    Fill buffer 2
end while
Stop Kinect 2
transMat1 ← self defined transformation matrix
transMat2 ← self defined transformation matrix
finalTrans ← 4x4unityMatrix
bestScore ← ∞
while bufferSize1 > 0 and bufferSize2 > 0 do
    temp1 ← pop(buffer1)
    temp2 ← pop(buffer2)
    temp1 ← noiseRemoval(temp1)                ▷ Optional
    temp2 ← noiseRemoval(temp2)                ▷ Optional
    temp1 ← downSample(temp1)                  ▷ Optional
    temp2 ← downSample(temp2)                  ▷ Optional
    aligned1 ← transformPointCloud1
    aligned2 ← transformPointCloud2
    ICP(aligned1, aligned2)
    if ICPscore < bestScore then
        bestScore ← ICPscore
        finalTrans ← ICPtransformation
    end if
end while
```

4.9.4 Depth measurement test on aligned Kinect data

For this test the Kinects was placed as shown in figure 4.7a on page 44 and algorithm 2 was used to align the streams from the two Kinects, defining a new origin between them, where a thought robot might be placed. Matrices

(4.10) and (4.11) were used as *transMat1* and *transMat2* respectively. The maximum buffer size were set to 100 and noise filtering were applied by using statistical outlier removal. The data was downsampled by dividing the 3D space into cubes of $1x1x1cm$ and defining a new point at the average position of all the points within these cubes. After the alignment, both the Kinects were activated and 100 frames from each were buffered. Then the buffered images were aligned and a rectangle of size $20x10cm$ from the new center of view, perpendicular to the z-axis, were filtered out. All the data from this area were written to file and further processed in MATLAB.

The test was run three times, for three different distances between the new transformed origin and the object in sight. This was to see if the object's placement relative to the Kinects' intersection point had any impact on the result.

Chapter 5

Kinect test results

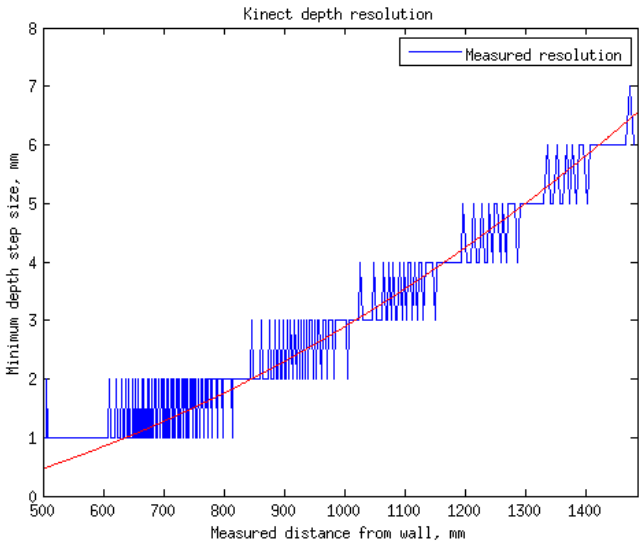


Figure 5.1: Kinect depth resolution as a function of the distance. The red line is the resolution found by [46], see equation (2.5) section 2.9

5.1 Resolution

Here follows the results from all the tests described in chapter 4.

5.1.1 Depth resolution

In figure 5.1 the measured depth resolution is plotted. The red line in the plot is the depth resolution estimated by [46]. The measured data seems to match the predicted resolution. We can see that in the closest operative range, $50cm - 60cm$, there is a resolution of $1mm$. From $60cm - 80cm$, the resolution switches between $1mm$ and $2mm$. As the distance goes beyond $81cm$, the best resolution one can hope for is $2mm$. One can also see that as the distances increases, the intervals before the quantization steps increases gets smaller.

Improving the depth resolution

Figure 5.2 on the facing page shows how the depth resolution step size decreases if we take the average depth from a $1cm^2$ instead of just a single pixel. We see that we now can achieve a depth resolution of less than $1mm$ at distances beyond $1.5m$, which is a great improvement from the $6 - 7mm$ resolution at the same distance with using only the data from one pixel.

5.1.2 XY resolution

Figure 5.3 on the next page shows the number of data points pr cm^2 captured in the $20x20cm$ square plotted against the theoretical number of pixels from figure 2.2c. The samples are taken in the range of $0.5m$

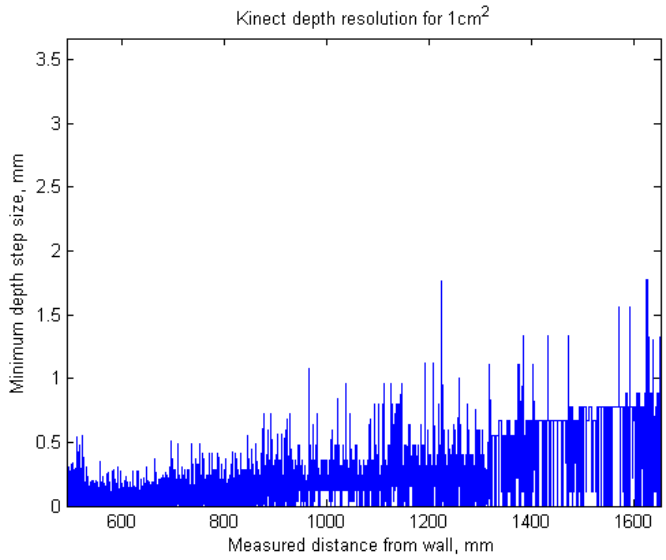


Figure 5.2: Kinect depth resolution from the average depth within a $1cm^2$ square as a function of the distance

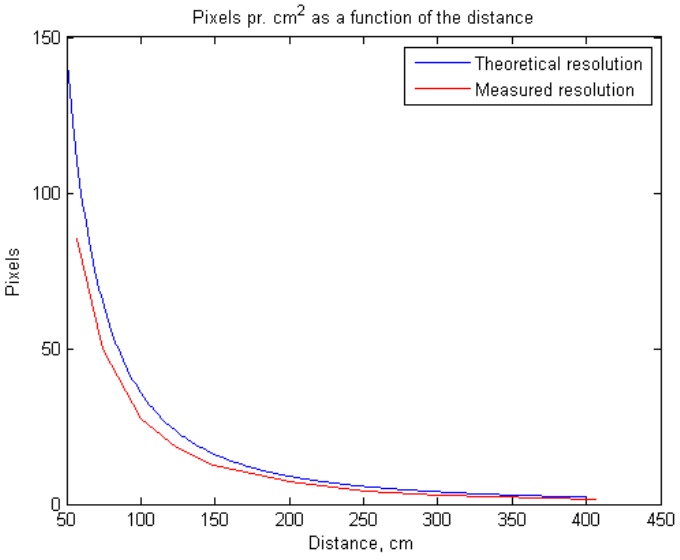


Figure 5.3: Number of pixels pr cm^2 in the 20x20cm square as a function of the distance, plotted against the theoretical resolution

– $4m$, and we can see that curve with the measured resolution nicely follows the theoretical resolution, but that it constantly is a bit lower than the theoretical maximum. This is probably because some of the IR light isn't returned to the sensor, or that the IR pattern isn't symmetrical as we assumed in the calculation of the theoretical resolution .

5.2 Stability

5.2.1 Time stability

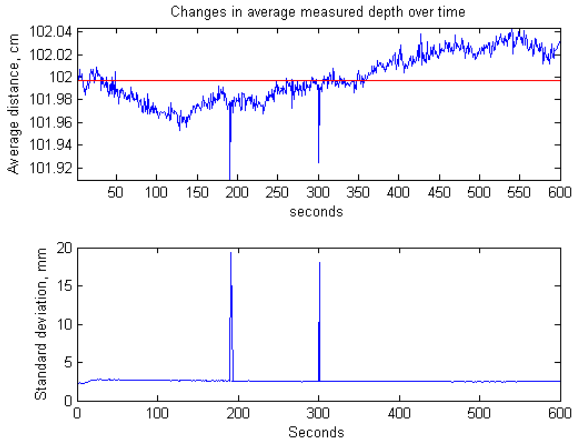
The plots in figure 5.4 on the facing page shows how the standard deviation and the average measured depth to the square on the wall changed over time. The red line in the graphs are the median value, added to help visualize the fluctuations in the results.

In figure 5.4a and 5.4b are the results from the 10 and 20 minute runs. We can see that there is a small variation in the average depth, with an amplitude of around $1mm$. In these two first graphs the variation is minor, and it may even seem to be periodic. But if we look at figure 5.4c, which has measurements for every second for an hour, we can see that the average measured depth increases more than $3mm$ from start until end. It seems like the depth evens out after around 30 min, but that there is more random noise (the sudden drops) in the measurements as time goes by. It is hard to tell what this increase of distance comes from, but it may have to do with temperature variations in the Kinect. The tests was taken directly after each other, in the order they are shown in the figure. When the Kinect is active it produces a lot of heat, so the 10 minute test was with a "cold" Kinect, while the last one was with a "warm" Kinect. It is possible that the temperature changes may slightly affect the IR wavelength and disturb the depth readings over time. To find out if it is a correlation between the temperature and the change in depth measurements it can be a good idea to run this test again and monitor the Kinect's temperature while the test is running. The standard deviation kept stable through the test runs, except for the frames where the average depth had sudden drops.

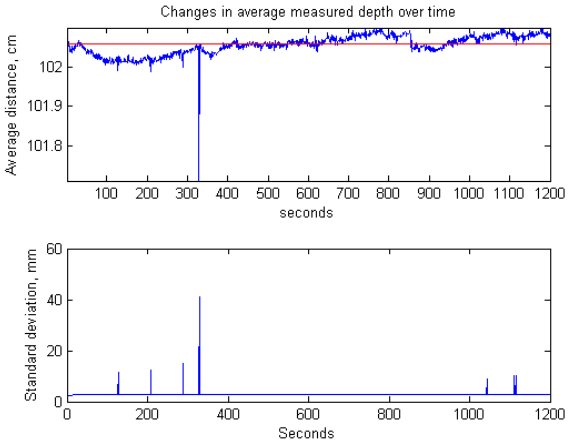
The reason why the 60 minute run starts at a lower value than the other two runs is that Kinect was moved between the second and third run and then slightly misplaced when put back.

5.2.2 Image stability

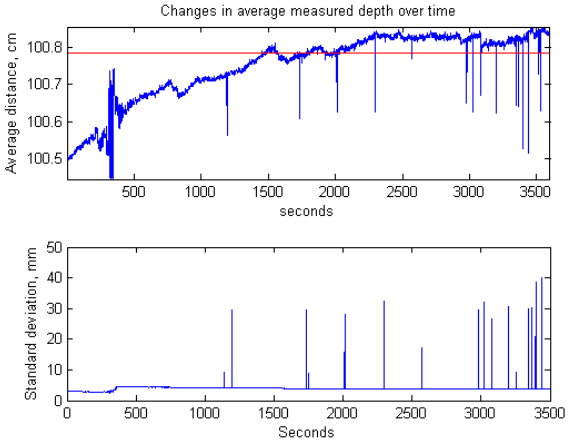
In figure 5.5 on page 54 we can see images that show 100 frames from the Kinect where the number of invalid measurements at each pixel is counted up, and forms the image. Figure 5.5a shows 100 frames taken against a flat wall, with the Kinect approximately $54cm$ away from the wall. The bar on the right of the image indicates the number of invalid pixels, showing that dark blue is 0 missing values, while dark red indicates that all the 100 values are missing. Whats interesting with this result is the frame of missing measurements that goes around 3 of the 4 edges of the image. The Kinect should deliver $640x480$ depth readings, but for some reason a lot of



(a) Average depth over 10 min

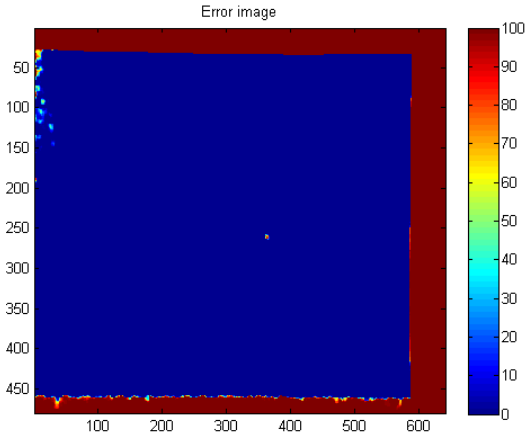


(b) Average depth over 20 min

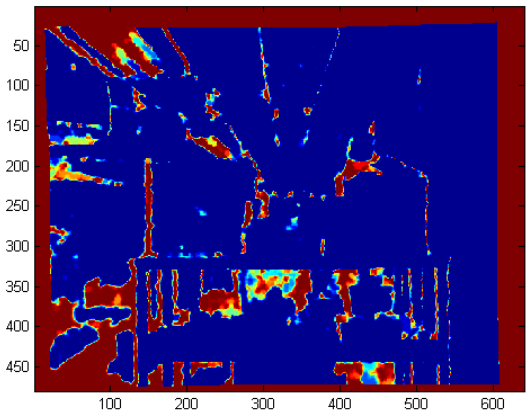


(c) Average depth over 60 min

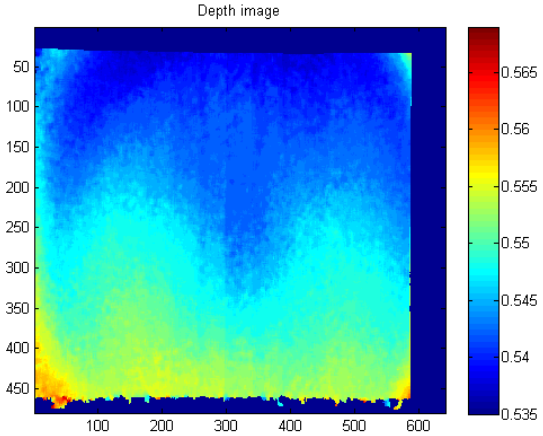
Figure 5.4: Average measured distance between Kinect and wall over time



(a) Invalid pixels in flat image



(b) Invalid pixels in office image



(c) Average depth to flat wall

Figure 5.5

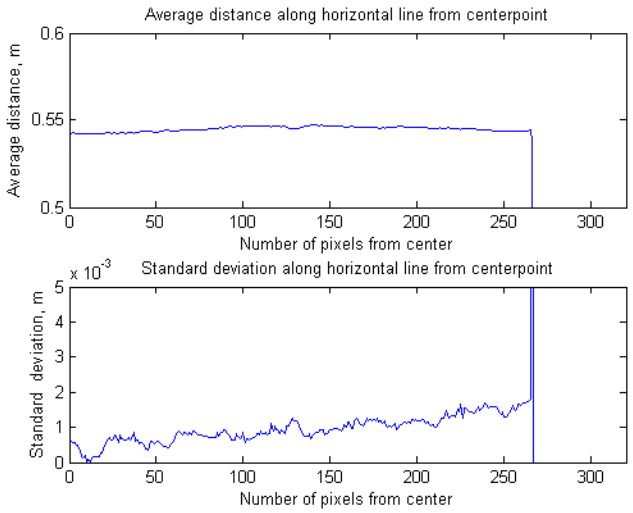
values are missing around the edges. There is also a small area almost in the center where there are a lot of missing values. These missing center pixels can come from dirt on the IR camera lens or emitter, it may be something on the wall that disturbs the IR reflection or it may be an error in the CMOS chip in the IR camera inside the Kinect. It may even be that the Kinect is placed to close to the wall, and that this disturbs the results. It has been noticed that when the Kinect gets to close to an object, it is the center pixel values that are lost first.

In the image in figure 5.5b, the Kinect is looking out over an office while the number of missing pixel values is counted for a 100 frames. Here there is a lot more missing values in the scene, but this comes from the varying placement and distance to the objects in sight. Some values are lost because of shadows, some are to fare away for the sensor to register them while other does not reflect the IR light back to the sensor. What is interesting to notice in this image is that the frame of invalid pixels around the image now is at all the 4 edges, and that the frame seems to be thinner or shifted to the right. One theory on why these edges appear is that in the edges of the Kinect's field of view, the IR light emitted from the Kinect may be at an angel that doesn't give a good enough reflection back to the Kinect. The Kinect marks the values as bad values it if is not sure what the real result can be, hence we may get a lot of missing values in the edges. But this does not explain why the missing pixels are around 3 edges in the first case, and 4 in the last case. But this edge of missing pixels indicates that the calculation on the Kinect's field of view, that was made in section 2.8.2, are slightly wrong. The actual field of view is a little bit smaller than expected. This is not a big problem, but it is good to know that this happens.

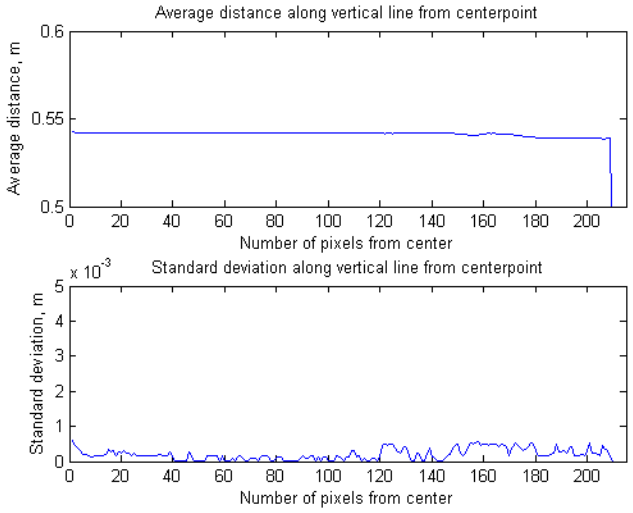
It can also be noted that the spot with missing pixel values in the center of image 5.5a of the flat wall, is not present in the image of the office scene. This may indicate that the spot appeared because the Kinect was placed to close to the wall, or that there was something on the wall that didn't reflect the IR light.

In figure 5.5c on the facing page we can see the average measured distance to the wall in the same 100 frames used to make figure 5.5a, the values in the bar on the right is given in meters. Since the wall is flat the distance to it should be the same over the entire image, but we can see that in these result the upper part of the image is measured to be a bit closer than the lower part of the image, while the distances measured from left to right seems to be constant. One reason for the different values from top to bottom may be that the Kinect was not placed entirely perpendicular to the wall. The Kinect's built in motor was used to tilt the camera to look directly at the wall, but it may be some inaccuracy in the motor or the accelerometer that gives a slight displacement of the camera angle. There is also some slack in the joints holding the sensor that may displace the camera.

Figure 5.6 on the next page shows the average depth and standard deviation along a horizontal and vertical line from the center of the image and out to the top and right edge. We see that the average distance is very stable all the way from the center and to the edges. The most interesting part of these graphs are the standard deviation, these can give an indication



(a) Average distance measured in the pixels along a horizontal line from the center pixel and to the right edge



(b) average distance measured in the pixels along a vertical line from the center pixel and to the top

Figure 5.6: Average distances along horizontal and vertical lines from the center of the depth image

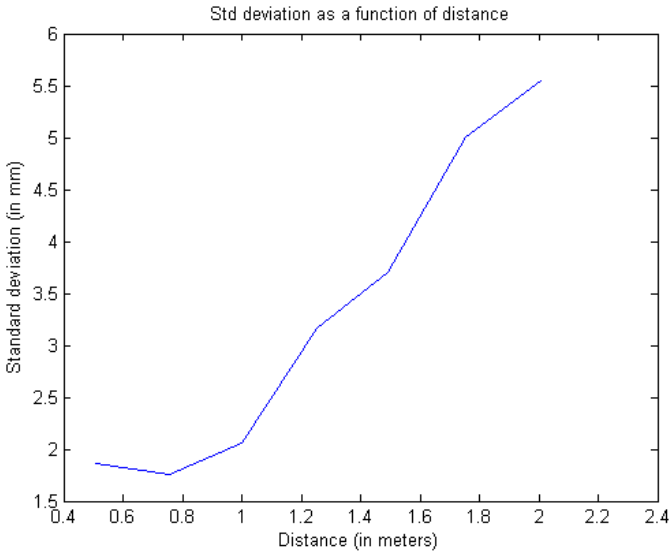


Figure 5.7: Standard deviation in measured distance from the Kinect to a 20x20cm flat square

of the variation in the measurements. Especially in the vertical line the standard deviation is very low, $< 1mm$. Along the horizontal line there seem to be a small increase of deviation as the distance to the center increases, but for both lines we can say that readings are relatively stable and that the entire depth image from the Kinect seems to give good depth readings. Maybe except for the corners where the depth seems to twist a little closer.

5.3 Measurement precision

Figure 5.7 shows the standard deviation in the average measured distance to the $20x20cm$ square on the wall. The deviation is low, $\leq 2mm$, up until $1m$. After $1m$ there is a dramatic increase in the deviation. So from this it may seem that if you want good precision in the measurements, you have to operate in the range of $0.5m - 1m$. This result is slightly better, but seems to be a a good match to the precision presented in [27].

5.4 Accuracy

5.4.1 Depth accuracy

Figure 5.8 on the next page shows the average measured distance between the surfaces, the standard deviation and the average error for the 100 frames for 7 different distances between the Kinect and the model. We can see that there is an error in the measured depth difference of $0.4mm - 2mm$ which is an error of $1\% - 4\%$. The test has been performed at to few distances to say for certain, but it seem like it is just as probable that the

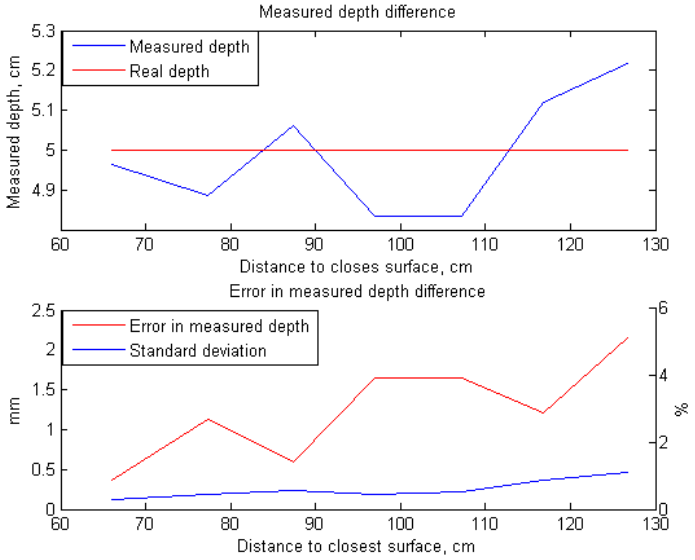


Figure 5.8: Average measured depth difference

measured distance is to long then to short. Also it is clear that the total error increases with the distance.

5.4.2 XY accuracy

As expected, the precision decreased as the distance between the paper and the sensor increased. But as the distance increased, also the accuracy decreased. This is in contrast to the expectation that the accuracy would be about the same, while the precision decreased with the increasing distance. The accuracy decreased from an error of about $5mm$ at the closest measurement and up to $50mm$ at $4m$ distance. At $1m$ distance the error is $\approx$ 5%, and already at $1.5m$ the error is $\approx$ 10%. It should be noted that this test only based its measurements on the points on straight lines from the center point and is very sensitive to the noise at the edges.

5.5 Edge accuracy

The graphs in figure 5.10 on page 60 shows the average distance and standard deviation from each pixel row in figures 4.3b on page 36 and 4.3c. Here we can see how the standard deviation increases at the location of the sharp edge while it is more stable along the tilted edge. In appendix D the graphs from all the eight test runs with different distances are shown. Notice that on the run with shortest distance between the Kinect and the model, the tilted edge was to close to the Kinect, so the data is corrupted by a lot of missing depth values. From these results we can see that on sharp edges you have to expect that it takes 2 - 3 rows of pixels to detect the edge and that you will get deviation peaks at the edge locations. For the tilted surface there is no major deviation peak at the edges, but there is generally

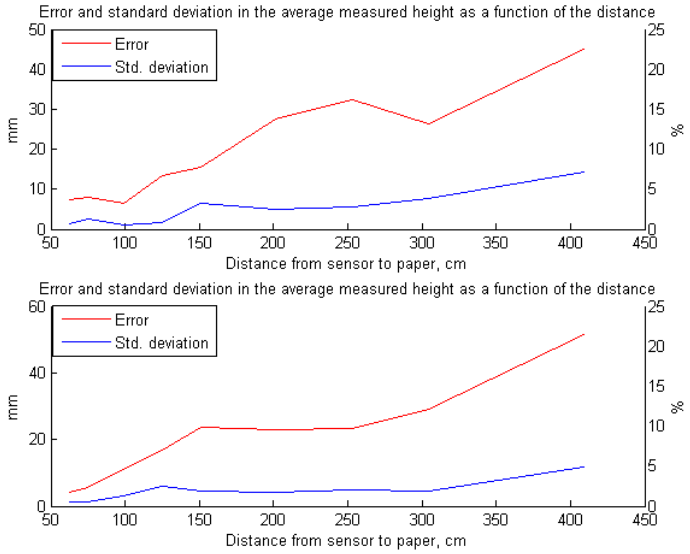


Figure 5.9: Error in measured width and height of an A4 paper attached to a window

a bit more deviation along a tilted surface than on a flat surface. From this we can state that the Kinect's accuracy drops around edges and that there is more noise at tilted surfaces compared to flat surfaces.

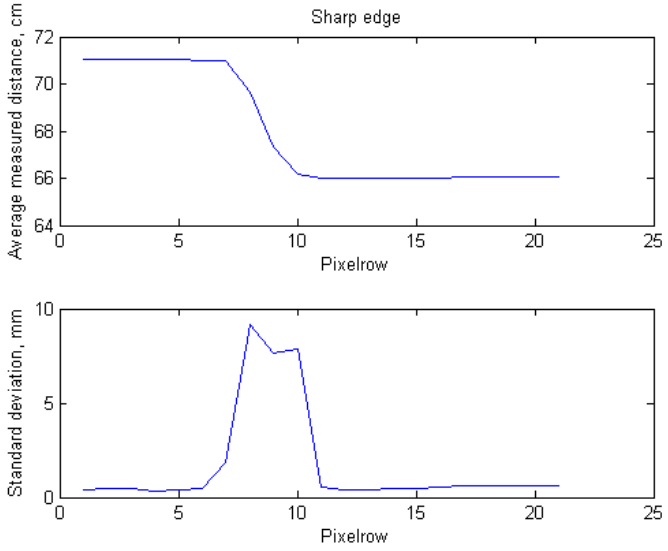
5.6 Model reconstruction

5.6.1 Convex sphere

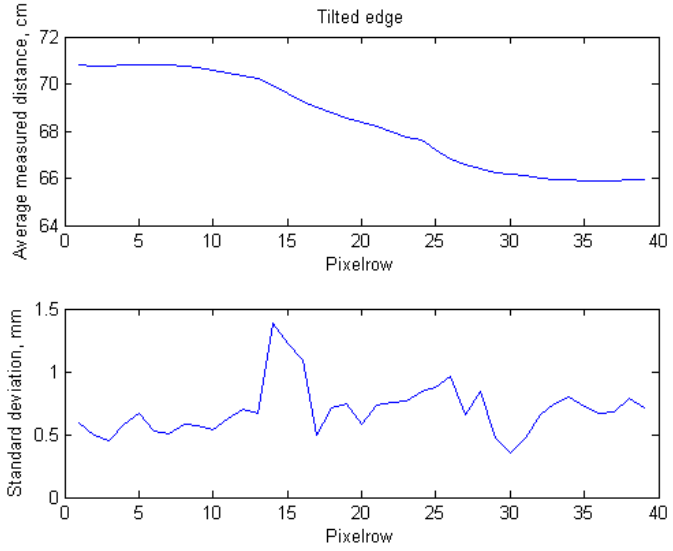
The calculated sphere radius at different distances can be seen in figure 5.11 on page 61. The red line indicates the real radius of the sphere, while the blue line is the calculated radius. For each distance the radius was calculated for a hundred frames, and the mean radius found is reported. As we can see from the standard deviation, the first measurement at 55cm is very unstable, while the measurements from 65cm to 125cm are stable. This is because the first measurement was at the lower limit of the Kinect's operative range, and thus there was a lot of noise in the readings, while at the other measurements the readings was stable and fine. The best result is at 65cm, where the calculated radius is 4.93cm which is only 0.7mm away from the real radius.

5.6.2 Concave sphere

Figure 5.12 on page 61 shows the results when estimating the radius from the concave side of the sphere. The results here are much better than in the previous test with the convex side of the sphere. But as explained earlier, the tests are done with different setup and it is therefor hard to give a direct comparison between the results.



(a) Sharp edge



(b) Tilted surface

Figure 5.10: Average depth and standard deviation in pixel rows in figure 4.3b and 4.3c

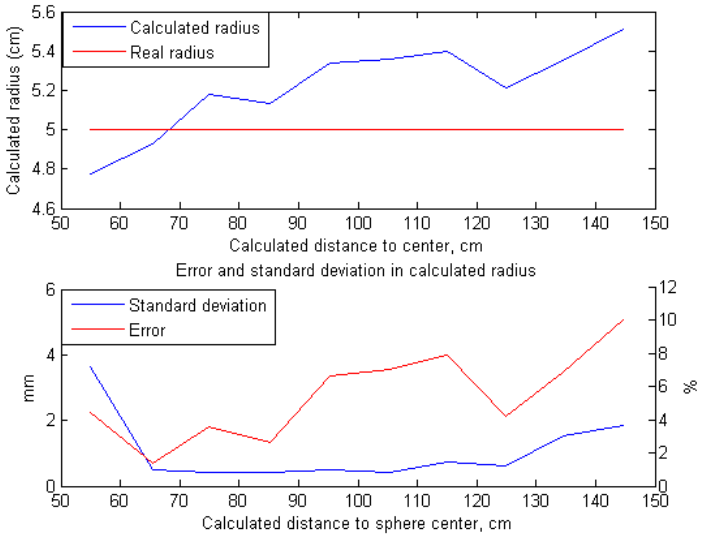


Figure 5.11: Calculated radius from convex side sphere

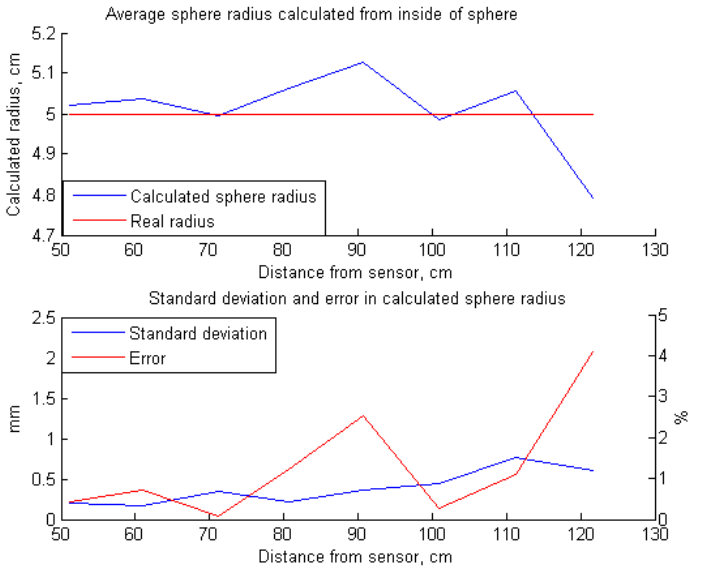


Figure 5.12: Calculated radius from concave side of sphere

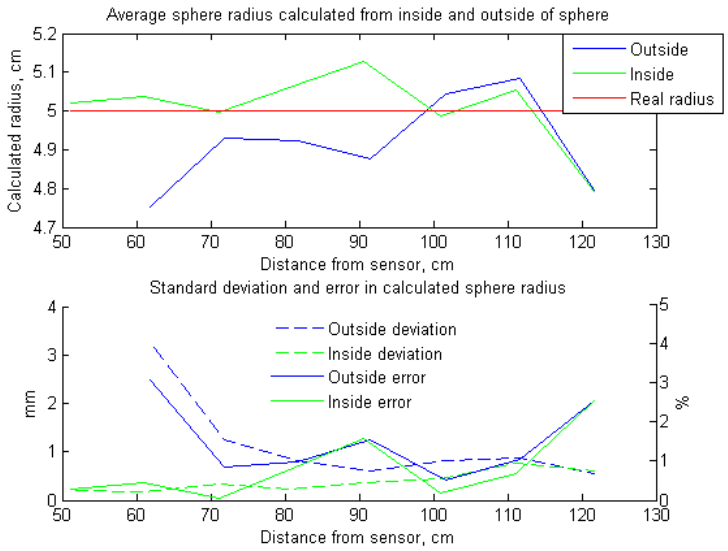


Figure 5.13: Comparison of calculated radius on data from inside and outside of a sphere

5.6.3 Comparison of convex and concave surface

When using the same setup for the full sphere as for the half sphere we get the results shown in figure 5.13. Here the estimation of the radius from convex side is plotted along with the plot from figure 5.12. First we can notice that there is one less measurement for the full sphere than for the half sphere. This is because with the inside of the sphere it was possible to take measurements closer to the Kinect than with the outside. There is also a lot less deviation in the measurements taken inside the sphere. Finally we observe that the radius calculated on the data from the inside seems to be more correct than the outside calculations. So from this it may seem like the Kinect gives better results on concave surfaces then on convex surfaces. It is also possible to take closer measurements of concave surfaces, compared to convex surfaces.

5.7 Angle of sight

In figure 5.14 on the next page we can see the number of data points the Kinect was able to capture from the sphere (dark blue curve). Based on the curve in figure 2.2c on page 19 we can see the theoretical maximum (red line) of how many pixels the Kinect should use on a sphere with a $5cm$ radius. The light blue line in figure 5.14 on the next page is the curve of a sphere that is best fitted to the number of pixels the Kinect really used. This sphere has a radius of $4.4cm$, indicating that the Kinect only saw about 88% of the sphere. The estimated radius of $4.4cm$ can then be placed into equation (4.2). This gives us $\alpha = 61.64°$. So from this we can assume that if α gets any greater than this, the Kinect will not be able to read any depth

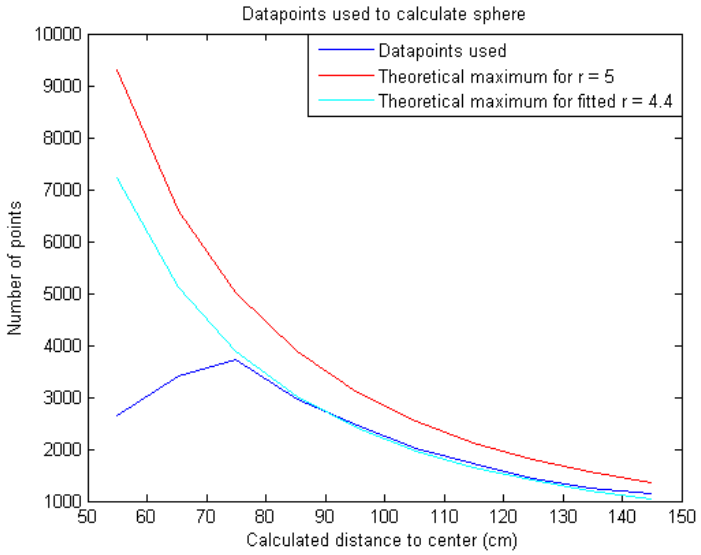
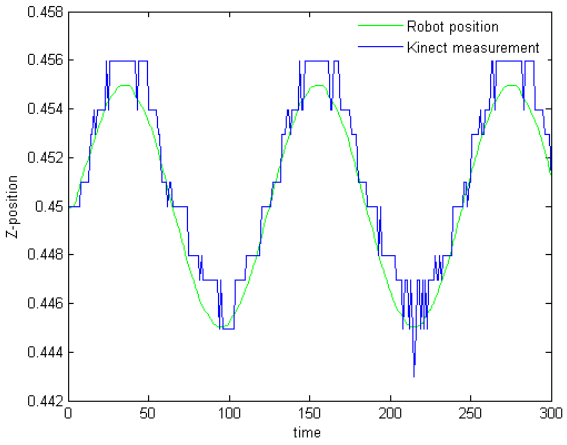


Figure 5.14: Estimate of how much of the sphere the Kinect could see, based on the number of data point captured from the Kinect.

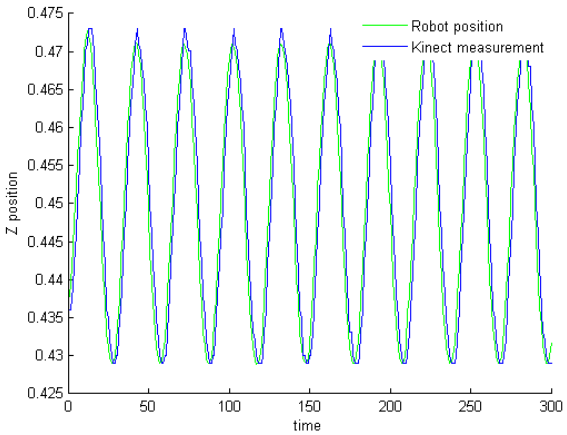
information from the object. But as we saw in figure 5.3 on page 51, even from a flat surface we got fewer data points back than expected from the theoretical resolution. So in reality the Kinect probably sees more of the sphere than the 4.4cm estimation, thus giving α a slightly bigger value than the one found here.

5.8 Kinect's frequency response and bandwidth

Figure 5.15 on the next page shows the position measured by the Kinect, compared to the real position of the robot tool. Calculating the actual delay between the real and measured position is hard since the Kinect data is unstable because of the poor resolution. This is especially a problem at low frequencies where the robot moves slow and at the lowest amplitudes. As can be seen in figure 5.15a the measured distance jumps up and down as the tool position moves between two depth quantization steps. Because of this there is often no single peaks or zero-crossings that can be used to measure the delay between the actual position and the position reported from the Kinect. The delay is defined as the time it takes from the robot crosses the 0.45 line and until the depth measured from the Kinect crosses the same line. In situations where the 0.45 line is intersected several times by the Kinect data, the delay is calculated as the average time between the signal first reached the 0.45 line and the last time it does so. The gain is calculated as the Kinect signal's peak-to-peak amplitude compared to the robot position's peak-to-peak amplitude. In figure 5.15a we can see why the peak-to-peak amplitude is used instead of the peak amplitude. In the first period in this graph, we see that the top points of the measured position is a lot higher than the real position, while the lowest measured positions



(a) f = 0.25Hz, A = 3cm



(b) f = 1Hz, A = 3cm

Figure 5.15: Kinect tracking of robot tool.

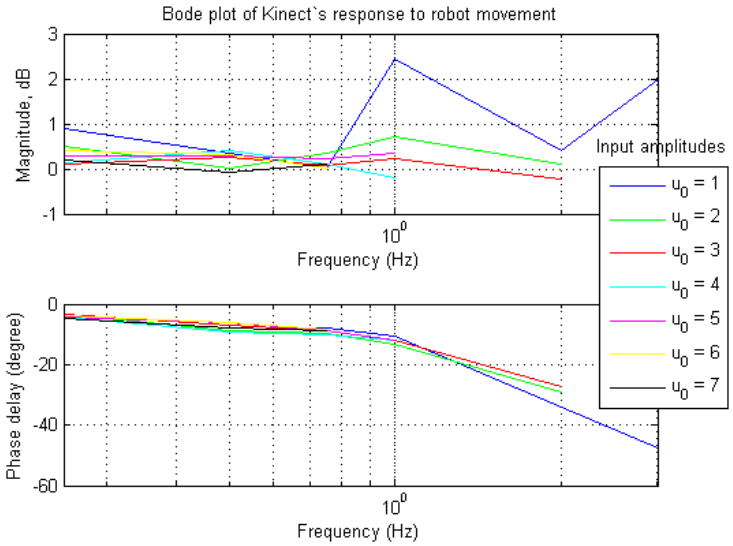


Figure 5.16: Bode plot of the Kinect's response to the robot's movement. "u_0" is the peak-to-peak amplitude the robot was told to follow.

is approximately the same as the real position. This means that the gain would be different if it was calculated from the highest peaks compared to the lowest peaks. During the tests there has been a tendency that the upper peaks would give a higher gain than the lower peaks. But there has also been results where both the high and low peaks are greater than the peaks in the real movements. This can be seen in figure E.1 in appendix E on page 119. That the high peaks often gives a higher gain than the lower peaks could indicate that the Kinect mostly measures a to long distance than a to short one. It could also be that the alignment between the Kinect measurements and the robot isn't accurate enough, but when the Kinect measured the robot position before it started to move the average error in the measurement was $< 0.1mm$, so this is not enough to cause such a high gain.

The frequency response from the Kinect's tracking of the robot's position can be seen in the Bode plot in figure 5.16. Note that the amplitude u_0 given in the plot is the peak-to-peak amplitude of the robot's control signal. As will be shown in chapter 7, the robot motion's amplitude is damped in relation to the input signal, so the amplitude the amplitude of the motion is a little lower than the amplitudes given in the graph.

From figure 4.6 on page 42 the gain was expected to decrease as the frequencies increased. This is clearly not the case. The ideal response would have been a gain of $0dB$, but in most cases the gain is $> 0dB$. The reason for this is the Kinect's resolution, giving inaccurate measurements. Because of this, and since the maximal test frequency was limited to $3Hz$, it is impossible to say anything about the real bandwidth of the Kinect, but it seems like the Kinect can track faster frequencies than the robot. This is based on the fact that the Kinect had no problem with tracking the robot at

	Received data points			
	Lower Kinect		Upper Kinect	
	Average	Std. dev.	Average	Std. dev.
Both active	14 496	74	14 756	171
Lower covered	374	251	15 345	134
Upper covered	15 167	65	89	158
With cardboard plate				
Both active	15 046	53	15 112	53
Lower covered	14	54	15 720	76
Upper covered	15 829	53	23	85

Table 5.1: Average number of valid depth measurements from two Kinects with 0° angle

its maximum frequency. The phase delay is relatively low between $0.25Hz$ and $1Hz$, but between $1Hz$ and $3Hz$ it increases fast. In time, the average delay between the robot position and the position measured with the Kinect is $0.0399s$. This is very good considering that the Kinect's frame rate of $30Hz$ would indicate a delay of $0.033s$. If we also withdraw the potential delay of $1/125s$ from the robot's position update, we get a total delay of $0.0399s - 1/30s - 1/125s = -0.0014s$. Which is clearly not possible since the Kinect cannot be ahead of time. But this just indicates that the delay between the robot's position update and the Kinect's measurement was not fully $1/125s$. Another reason why the Kinect seems to know the robot's position before it gets there is the Kinect's poor resolution compared to the robot's. Because of the Kinect's resolution at this distance it may report that the robot is $2mm$ closer or further away than it really is, and this can make it seem like the Kinect is ahead of the robot.

Finally it should be noted that all the data from the Kinect is based on the readings from a single pixel and is thus more attractive to noise than if the depth had been measured from a weighted sum of several pixels. It was also shown in section 5.1 that the Kinect's depth resolution increases if the measurements are taken from a number of pixels. The response might have been better if this had been done.

5.9 Multiple Kinects

5.9.1 Interference

The results from the interference test can be seen in table 5.1. The first thing to notice is that the number of pixels captured by one Kinect is higher if the other Kinect isn't emitting any light. So one Kinect is disturbed by the other when they are both emitting light on the same surface, but the difference is far from as large as expected. The upper Kinect received 4.6% more data points when it was the only Kinect emitting light, compared to when they both were emitting. Similar the lower Kinect received 4% more data. One

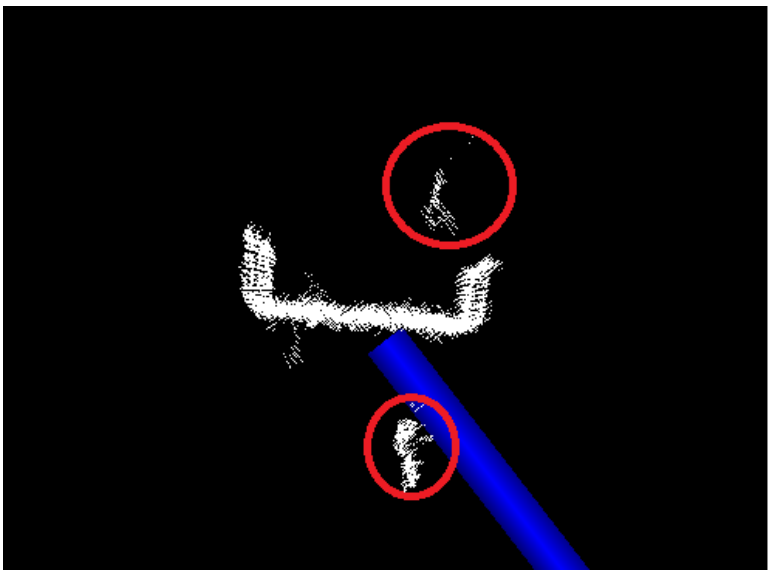


Figure 5.17: False data readings caused by Kinect interference

of the most interesting results is that the Kinect which has the blocked IR emitter still captures some depth readings, so one Kinect can clearly read the IR light emitted from another Kinect. The amount of captured data is relatively low, so it is still a pretty good result. One example of depth readings that appear out of nowhere is shown in figure 5.17. Here we see two aligned images of a box, the noise indicated by red circles are noise that appears when both Kinects are active. The blue object in the image is the z-axis of the "main" Kinect.

Further we see that all the results improved with the cardboard plate placed in front of the box, compared to just the white floor. Especially the amount of captured data from a Kinect with blocked IR emitter has been greatly decreased. This is probably because of that the floor is very reflective and the light that should have been reflected from the floor and back to the Kinect[1], is sent on to the the box before it returns to the Kinect.

Table 5.2 on the next page shows the results from the test with the Kinects placed with a 90° angle between them. Like before we can see that one Kinect performs better when the other's IR emitter is blocked. What is totally different in these runs is that there seems to be poorer performance when the cardboard plate is placed in front of the box. The Kinect with the blocked emitter captures less data from the other Kinect, which is good, but the Kinects also receive less of their own data when the plate is in place. One reason for why the results are worse now, may be that since the plate now covers a smaller part of the surface in front of the Kinect's, the edges on the plate may generate more noise than the plate removes. So it is clear that there is no absolute solution to what setup is the best when it comes to the area around the object in sight. It is something that has to be considered

[1]It has been noticed during testing that a Kinect standing on a flat surface will never get depth readings from the surface it is standing on. This is probably caused by the angle between the Kinect and the floor.

	Received data points			
	Right Kinect		Left Kinect	
	Average	Std. dev.	Average	Std. dev.
Both active	8 392	80	9284	47
Left covered	9 511	31	6	43
Right covered	9	50	10 236	34
With cardboard plate				
Both active	8 294	66	9012	54
Left covered	9351	57	0	0
Right covered	1	10	10 031	33

Table 5.2: Average number of valid depth measurements from two Kinects with 90° angle

and adjusted to individual cases.

5.9.2 Depth measurement test on aligned Kinect data

Figure 5.18 on the facing page shows the results when the aligned depth images are used to take depth measurements [2]. The graphs shows the average measured distance to the $20x10cm$ rectangle for 100 frames, along with the standard deviation. The four plots in each graph represents the results with the transformed data from

1. Only Kinect 1 active
2. Only Kinect 2 active
3. Combined data from when the Kinects ran one at the time
4. Combined data from both Kinects active at the same time

As expected the standard deviation in the average depth is much higher, 80% - 100%, when both Kinects are active, compared to the data captured with only one Kinect at the time. And as before, in tests with only one Kinect, the deviation increases with the distance. The distance between the Kinect's and the object seems to be more important for the results than where the object is placed relative to the Kinects' intersection point. We can also note that the results indicate that the alignment of the data isn't perfect. With perfect alignment the average distance should have been approximately the same for all four measurements, but there is a small difference of $0.5mm$ – $1mm$ in the measured distance. So the alignment process may need further improvement.

[2]The captured data were corrupted by a lot of noise like the one showed in figure 5.17 on the previous page. To remove this in MATLAB all depth values outside (0.4m – 1.2m) along the z-axis were filtered out.

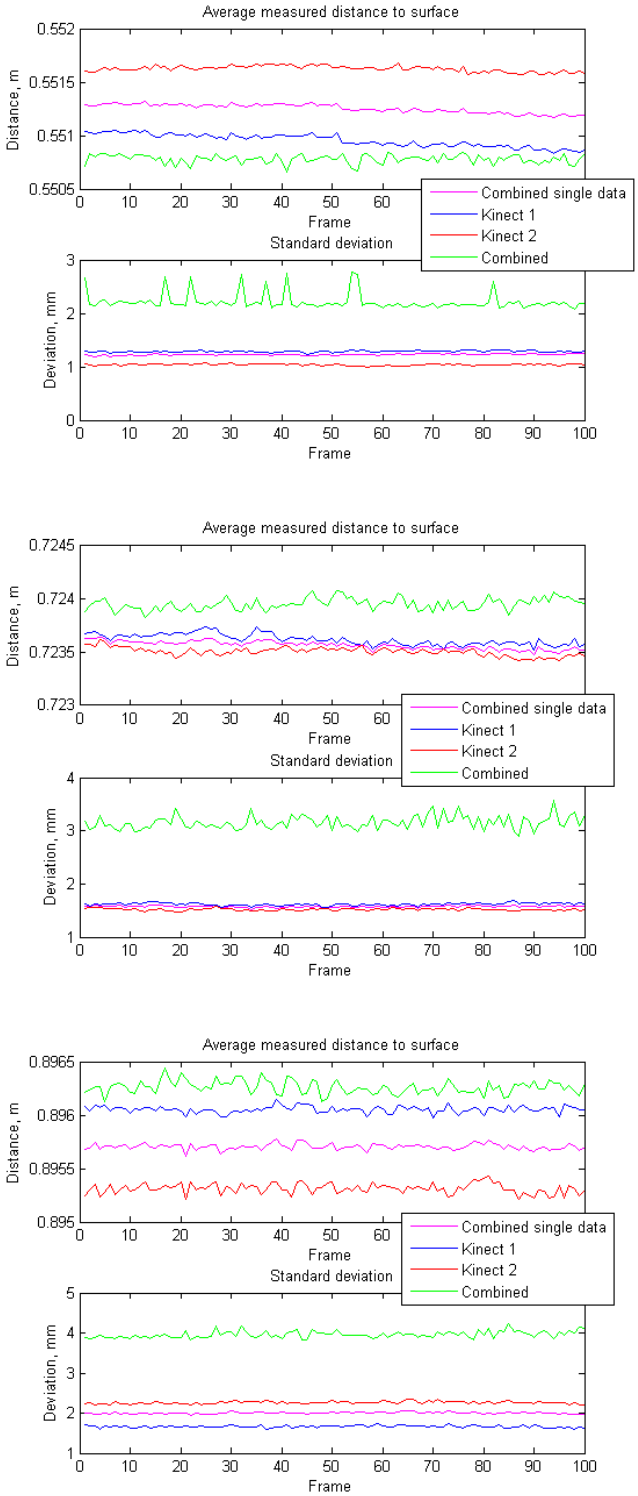


Figure 5.18: Average measured depth from two Kinects compared to average depth from single Kinect. "Combined single" data is the result from aligning the data captured with one Kinect running at the time. "Combined" is the algined data when both Kinect were active at the same time.

(a) Still standing object at close range

(b) Moving object at close range

Figure 5.19: Difference in valid depth measurements from a moving and still standing object at the same distance

5.10 Other observations

When objects gets to close to the Kinect, it is the depth readings from the Kinect's center of view that is first corrupted. It has also been noticed that this corruption is a bigger problem on objects that is standing still than on objects that has a small continuous movement. This can be seen in figure 5.19, in 5.19a the object is standing still at the lower limits of the Kinect operative range, while in 5.19b the object has the same distance to the Kinect but is constantly moved to the left and right. We can see that the difference in valid depth measurements is massive. This may indicate that the Kinect works better on moving objects than on still objects, which in our case is a good thing since we hopefully will be monitoring a breathing patient.

Part III

The Kinect–robot system

Chapter 6

Kinect–robot interface

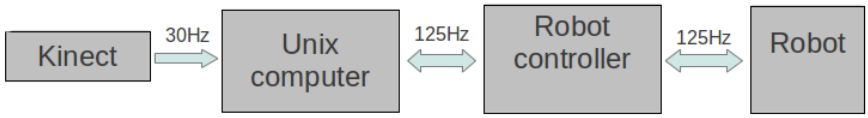


Figure 6.1: Model of the robot system with the Kinect

6.1 The robot

The robot in this project is a UR-6-85-5-A from Universal Robots, from now on just called Universal Robot or UR. The UR is a 6 degree of freedom manipulator that has been approved to work in human environments without any extra safety features. UR has maximum working velocity and force constraints, causing security stops if any of these limits are breached. UR has a reported maximum velocity of $\frac{180^\circ}{s}$ at each joint, which is approximately $1m/s$ for the end effector.

The UR has a real-time interface that can be used to connect the robot to a computer and control it from this. This interface with the computer gives the robot system two control layers. First there is a controller on the Unix computer that sends real-time velocity commands for each joint to the robot. Then the robot has an internal controller that sends a current to the motor in each joint, based on the velocity command. The robot controller sends data back to the Unix computer, these data describe the current position of the joints, velocity etc. This communication between the computer and the robot operates at $125Hz$. Figure 6.1 shows a simple illustration of how the system is set up. The Kinect will send 3D image information to the computer, where positional commands is derived from the images and sent on to the robot. The Kinect data are processed in its own processing thread that runs at $30Hz$. The Unix computer has additional input from a force sensor and an ultrasound probe, these are not included in the figure since they are not part of this thesis.

On the computer an operator can control the robot with a LabVIEW GUI. In LabVIEW all the control parameters for the robot is set, and one can give velocity, joint angles or position commands to the robot. The position commands are given in Cartesian coordinates, and describes where the robot's tool should be placed, relative to the robot base coordinate system. When using position commands, the robot tool orientation relative to the robot base has to be defined by the operator. From the position command, the inverse kinematics of the robot is used to calculate the angle needed on each joint to place the tool in correct position. For this robot there is eight potential different joint angle configurations that will place the robot in the correct position. In situations where there are several solutions, the solution that is closest to the current joint angle rotation are selected. The system uses a proportional controller to calculate the velocity commands for each joint.

Proportional controller

The robot velocity is controlled by a proportional controller [51, p.216]. So the velocity is given by the difference in the desired position, and the current position multiplied by a predefined gain (6.1). P_d and P_c is the desired and current robot tool position.

$$Out = (P_d - P_c) * Gain \tag{6.1}$$

This causes the robot to move fast when the error between current and desired position is large, and slower as the robot closes in on its target position. A large gain gives fast movement and better response in the system, but if the gain is to large the output may be unstable. So one have to set a gain that gives the best response to the input and still produces a stable output. It is also worth noting that with a proportional controller, there will always be an error between the desired position and the current position [51, p.136], unless the gain is set to ∞, which is clearly impossible.

6.2 Kinect – robot interface

The existing interface between the computer and robot have been the basis when making an integration between the robot and the Kinect. Most of the computer–robot interface is written in C++, so the additional Kinect interface is written as a simple C++ header file, based on OpenNI and PCL. Requirements for the interface to work are a working installation of OpenNI and PCL. By installing the prebuilt binaries from the PCL website all other dependencies are automatically included and installed.

The thought behind the interface is that new position commands for the robot will be derived from the depth map captured by the Kinect. The interface offers methods to start and stop the Kinect, translate points from the Kinect's coordinate system to the robot's coordinate system, and retrieve the latest position calculated by the Kinect. The control algorithm that finds the next position the robot should move to is up to the user to implement, so that he/she can get exactly the wanted functionality. The code of the interface can be seen in appendix B on page 107.

The Kinect uses a few seconds to start and to calibrate itself to give accurate depth measurements, so it is recommended that the Kinect interface is initialized at the same time as the robot system is turned on. To start the Kinect the *initializeKinect()* function should be used. This function sets all global variable to predefined values and opens a connection to the Kinect with the help of PCL's *openNIGrabber*. The PCL interface requires a callback function that new data from the Kinect can be delivered to. This callback function is *cloud_cb(const pcl::PointCloud<PointT>::ConstPtr & cloud)*. *PointT* tells what kind of point cloud PCL shall retrieve from the Kinect, in this case *PointT* is sat to PointXYZ, which gives data from only the depth sensor. *PointT* can also be sat to PointXYZRGBA, which will give depth data aligned to RGB data from the RGB camera on the Kinect. When the Kinect interface is set up, and the callback function is connected to the

Kinect process, the Kinect is started with the call *kinectInterface→start()*. This will start a new process that receives data from the Kinect at 30 frames pr. second. When a new frame is ready it must be sent to the *findNextPosition()* function where the user implemented control algorithm should be placed. To save processing power this control algorithm should find the next position in the Kinect's coordinate system and then call the *transformPoint()* function to transform this single point from the Kinect's coordinate system to the robot's base coordinate system. This saves some processing time since only a single point has to be transformed, instead of all the potential 307200 data points a single frame can consist of. The new position is saved by *transformPoint()* and then the robot controller can use *kinectGetLastPosition()* to get the latest position from the Kinect. The variable storing the next position is shared by the Kinect process and the robot controller process, so to avoid that both processes access this variable at the same time a *boost::mutex*[1] object is used. To stop the Kinect and close down the interface the function *stopKinect()* should be used. If the robot control process should try to read a new position command at the same time as the Kinect process stores one, the robot will be given the last found position.

6.2.1 Comments to the interface

The transformation matrix, *kinectTransMat*, used to transform data points from the Kinect's coordinate system to the robot base must be defined by the user. At the moment this has to be done in the source code in the *initializeKinect()* function. With the current configurations this matrix is defined as a unit matrix, so no rotation will be performed. A calibration process to find the ideal transformation matrix between the Kinect and the robot is yet to be developed. The *kienctRotation* array stores the rotation the robot's tool should have relative the the robot base. In the implementations used so far to control the robot with the Kinect, this rotation has been kept fixed. At the current setup the tool z-axis will constantly point the opposite way of the robot base z-axis. It should be implemented code that will also set the rotation of the tool to the desired position, allowing the rotation to change as the environment changes. If the tool rotation should be kept fixed at all time, a better solution than to set the rotation directly in the source code would be to let the user define the rotation in the LabVIEW GUI.

6.3 Kinect – robot system's bandwidth

The principle of frequency response and bandwidth was explained in 4.8. The following test intends to find the bandwidth of the Kinect–robot system. Instead of connecting the Kinect to the robot, a process that simulates the Kinect was added to the robot controller on the Unix computer. The process generates a sine wave and uses this to give position

[1] www.boost.org, boost is included in PCL

commands to the robot, the program will only change the z-position of the robot tool. The x- and y-position will not be changed during the test. The process will run at 30Hz, which is the same as the Kinect. Position commands $z(t)$ will be updated by taking samples from the sine at 30Hz using equation (6.2), where A and f is the amplitude and frequency of the sine and t is the time, counted as clock cycles in the simulated Kinect process.

$$z(t) = \frac{A * sin(t * 2\pi * f)}{30} \tag{6.2}$$

Like in the test of the Kinect's bandwidth, the frequency was gradually increased and for each frequency the test were run for increasing amplitudes. So the test algorithm was similar to the one showed in alg. 3. For each combination of frequency and amplitude the test ran 300 clock cycles in the Kinect process, which is the same as 10 seconds. The test starts with

Algorithm 3 Algorithm to test the robot's bandwidth

for $f = f_{min} \rightarrow f_{max}$ **do**
  for $A = A_{min} \rightarrow A_{max}$ **do**
    for $t = 0 \rightarrow 300$ **do**
      $nextRobotPos \leftarrow B + \frac{A*sin(t*2\pi*f)}{30}$
    end for
  end for
end for

the robot tool in a predefined default position at $(-0.5, -0.11, 0.45)$, so B is a bias added to the position command that lets the tool move up and down around the default position. This default position is also selected to guarantee that the robot has free space to move in and avoids collisions. For extra security a bounding box of size $20x20x20cm$ was defined around the robot tool, with center at the tool default position. If the robot tool reaches any of the limits in the bounding box, it will be stopped. It is only the z-axis coordinate that is changed during the test, the x- and y-coordinate remains the same, so the robot tool should move straight up and down for the entire test run. For these tests the gain in the proportional controller was set to 9. This value was chosen from visual inspection of the robot movement when the robot was given single position commands through the LabVIEW interface. When the gain was set higher than 9 the robot movement seemed to oscillate when it closed in on the target position. In addition to the gain, the maximum velocity command that the computer can send to the robot must be set. The maximum speed the robot can move with before it is stopped for security reason is π radians pr. second. So to avoid the security stop the maximum velocity was limited to 3 radians pr. second. The results can be seen in section 7.1.

6.4 Demonstration of how the Kinect directly controls the robot's position

This final test was a demonstration of how the robot's tool position can be directly controlled by the Kinect. The demonstration is designed to show how the Kinect can be used to track a surface and generate position commands to the robot so that it will follow the surface. The Kinect is placed below and slightly to the side of the robot tool. Then a cardboard plate, that is initially held at the same height as the robot tool, is slowly moved up and down above the Kinect. By taking depth measurements from the center pixel of the depth image received from the Kinect, position commands are sent to the robot. Just as in the test of the Kinect's bandwidth, the data from the Kinect and the robot's initial position is aligned by letting the Kinect first measure the distance to the plate when it is next to the tool.

The ideal for this test would be to have another robot. Then one robot could move, and the other could follow that movement with the help of the Kinect. Then the time and actual position of the movement could be stored along with the measurements from the Kinect and the position of the robot and the exact delay could be calculated from this.

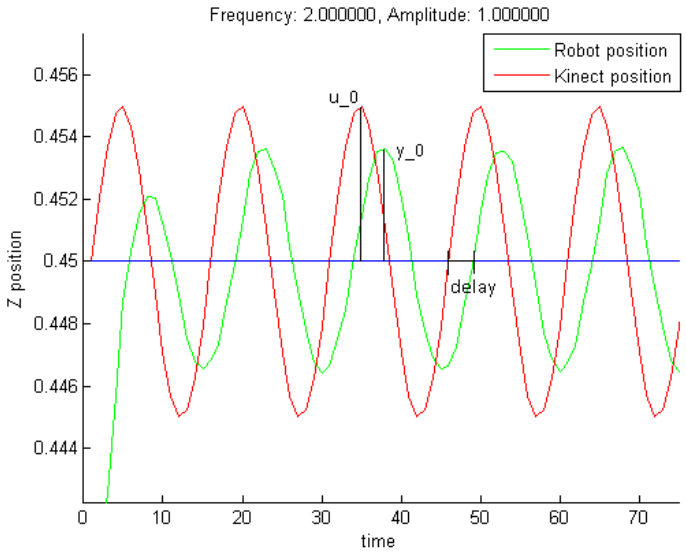
Chapter 7

Kinect–robot test results

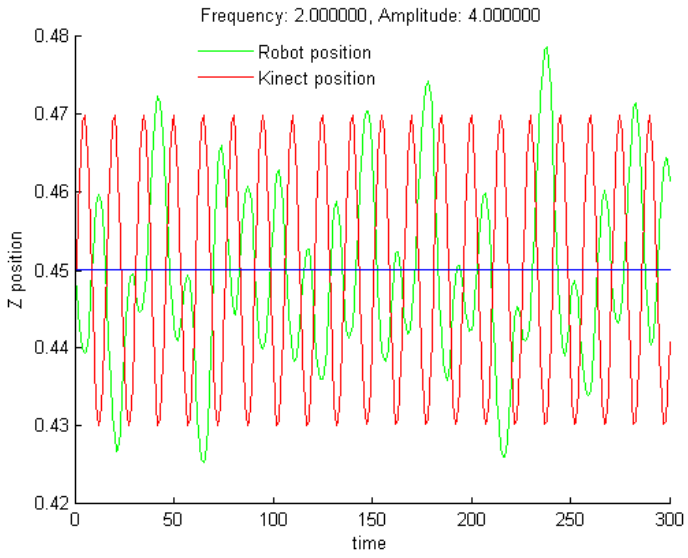
7.1 Kinect – robot system's bandwidth

Figure 7.1a on the facing page shows how the robot responds to an input with $f = 2Hz$ and $A = 1cm$. It is important to note that the amplitudes mentioned in the discussion about the robot system are given as peak-to-peak amplitudes. The control position is called *Kinect position* since it simulates position commands from the Kinect. The x-axis in the graphs represents the time, given as clock cycles in the Kinect process. In this graph we can see how the robot's position nicely follows the position commands, but that there are some delay and that the response is damped compared to the input signal. Figure 7.1b shows how the robot moves when the frequency and amplitude is to high for it to follow. The robot movement never stabilizes and it is clear that it can not operate at this combination of frequency and amplitude. Table 7.1 on page 82 shows the results from all the variations of f and A tested for. X means that the output never stabilized, as shown in figure 7.1b, while $-$ indicates that for this combination of f and A no test was performed. $|H(j\omega)|$ is calculated from the mean gain between all the top peak pairs, and $\angle H(j\omega)$ is the mean delay between every time the control position crosses the 0.45 line and when the robot position crosses the same line. All the values are calculated after the robot movement has stabilized. For $f = 3/4$ and $A = 10cm$, the robot reached the lower boundary of the security bounding box and the test was stopped. This means that when the robot stopped, the tool was in z-position lower than 0.35, and with a maximum peak-to-peak amplitude of $10cm$ it is clear that the robot had moved $5cm$ beyond its designated target. This is probably because the gain in the proportional controller is set to high, causing the robot to move to far and to fast, and thus move past the desired position before it had a chance to stop.

In all the tests, with different frequencies and amplitudes, the time delay in the system was very stable, mostly staying between 2/30s and 3/30s, with the highest delay at the low frequencies, and a total average for all the runs of $0.0952s$. It is natural that the time delay is a little less at higher frequencies, since the error between the desired position and the robot's actual position gets higher with higher frequencies. With an higher error, the output from the proportional controller increases, telling the robot to move faster. But even though the delay in time is smaller, the phase delay increases with higher frequencies. This can be seen in the Bode plot of the results in figure 7.2. In the Bode plot we can clearly see that the statement made by [51] is true. As the input frequency increases, the gain on the output decreases. For each frequency the magnitude of the response is mostly stable, independent of the given amplitude, but as the frequency increases there is fewer amplitudes that the system can deal with. Also the phase delay seems to increase with the amplitude. If we define the cut-off frequency of the system as where the magnitude crosses the $-3dB$ line then we can see that this happens at $\approx 2Hz$. But at $2Hz$ the maximal amplitude it manages to follow is $2cm$, so it is not a very good result. Since the robot tool has a reported maximum velocity of $\approx 1m/s$, the response was expected to be better. Of course one cannot expect max velocity with all the direction



(a) Stable system response



(b) Unstable system response

Figure 7.1: Stable and unstable system frequency response. Time is given as number of clock cycles in Kinect process

	$\lvert H(j\omega)\rvert = \frac{y_0}{u_0}$									
ω \ u_0 (cm)	1	2	3	4	5	6	7	8	9	10
1/4	99.3%	99.3%	98.6%	98.8%	98.9%	98.6%	99.5%	99.0%	99.1%	99.2%
1/2	96.9%	95.4%	96.1%	95.7%	96.1%	96.0%	95.6%	95.5%	95.8%	95.6%
3/4	90.3%	90.5%	92.0%	91.3%	90.8%	91.3%	90.7%	91.1%	91.4%	x
1	84.1%	84.7%	86.2%	86.9%	86.7%	84.9%	x	-	-	-
2	72.4%	68.6%	x	x	x	-	-	-	-	-
3	59.4%	x	x	x	x	-	-	-	-	-
4	x	x	-	-	-	-	-	-	-	-
	$\angle H(j\omega)$									
1/4	3	3	3	3	3	3	3	3	3	3
1/2	2.89	3	3	3	3	3	3	3	3	3
3/4	2.64	3	2.86	3	3	3	3	3.07	3.21	x
1	2.32	2.42	2.68	2.89	2.95	3	x	-	-	-
2	1.77	1.97	x	x	x	-	-	-	-	-
3	2.00	x	x	x	x	-	-	-	-	-
4	x	x	-	-	-	-	-	-	-	-

Table 7.1: Results from bandwidth test. The delay is given as average number of clock cycles in the simulated Kinect process. "X" means robot never stabilized, "-" means no test performed.

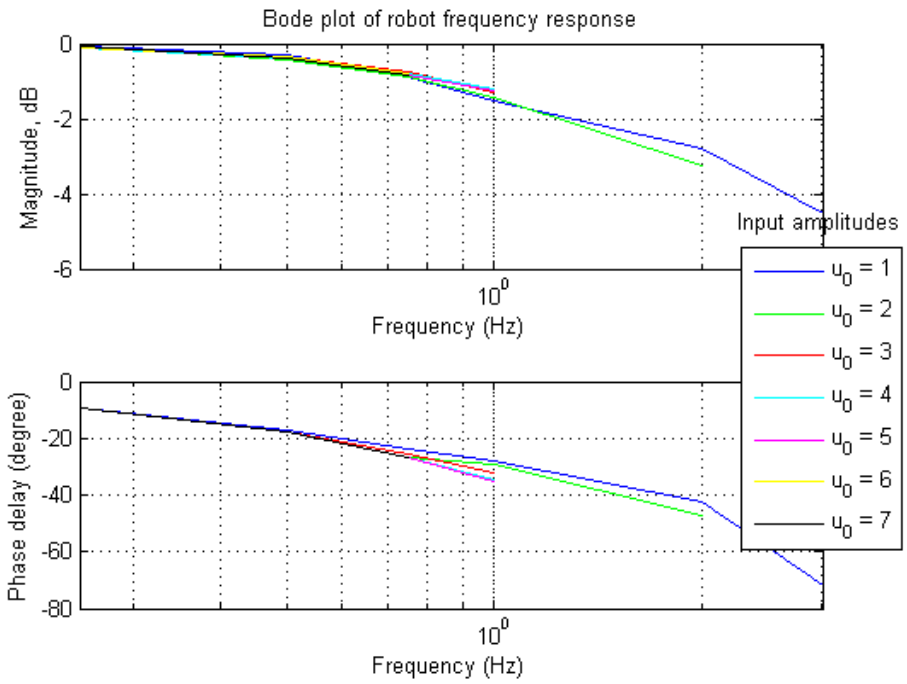


Figure 7.2: Bode plot of the frequency response, u_0 is the peak-to-peak amplitude (in cm).

changes there are when the robot is controlled with a sine wave. $1m/s$ is probably the maximum speed when moving in a straight path.

Effect of to high gain

As mentioned earlier, the gain in the proportional controller affects the response of the system. Figure 7.3 on the following page shows an example of what happens if the gain is set to high. The graph shows the robot's response when the control signal has a frequency of 1Hz and the amplitude changes from $7cm$ to $8cm$. In the time interval between 0 and 300 the amplitude is $7cm$. At first it seems like the system is about to stabilize, but as time reaches 200 we can see that the amplitude of the robot motion starts to increase, moving beyond the control signal. When time = 300, the amplitude of the control signal increases to $8cm$. We can now observe that for each period the amplitude of the robot movement increases, until it is so far away from the control signal that it gets security stopped since it breaches the bounding box.

Because of the delay between the control signal and the robot movement there will always be an error in the position of the robot. When the amplitude increases this delay increases as well. And since the robot velocity is controlled by this error the velocity increases. So the velocity becomes so high that the robot moves beyond its target position before it has time to stop. This increases the position error even more, and the velocity will never stabilize. With a lower gain the robot would probably be able to track

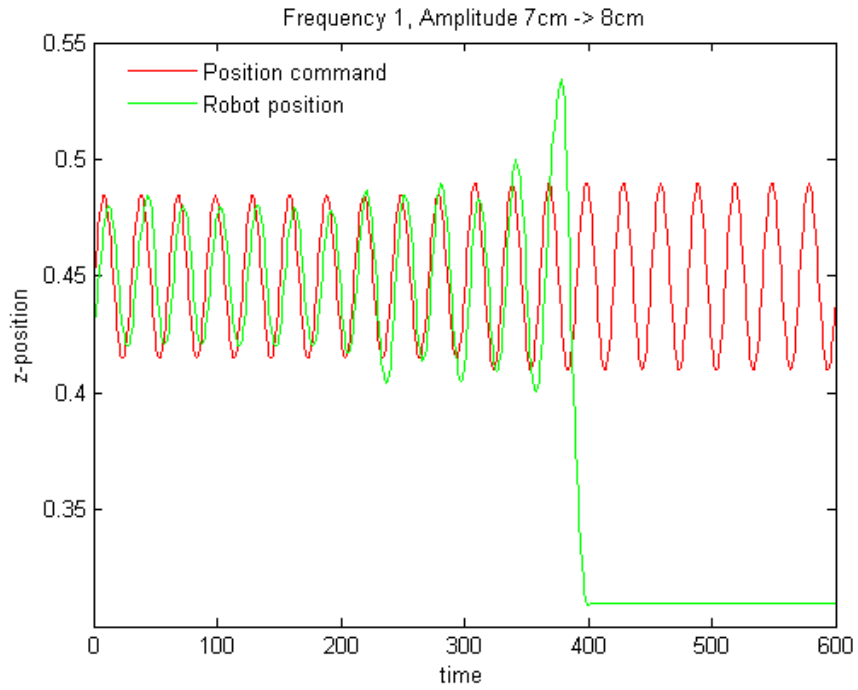
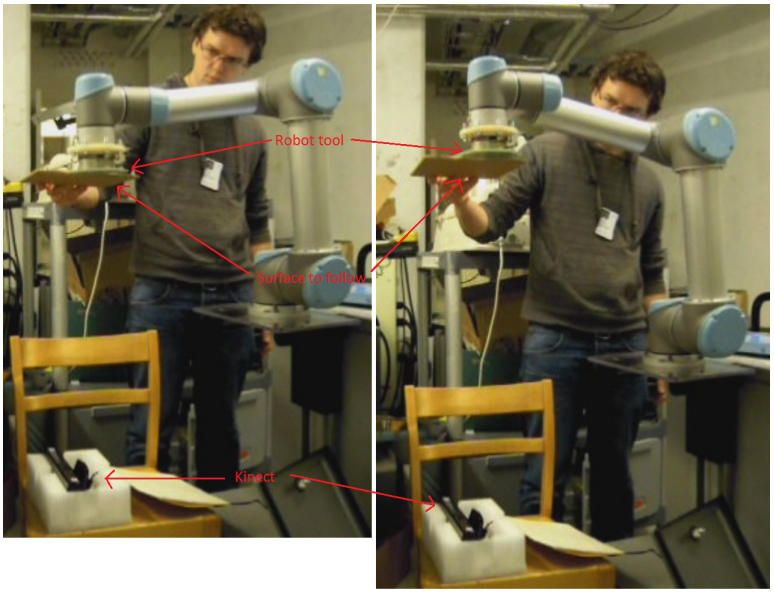


Figure 7.3: Robot movement with to high gain

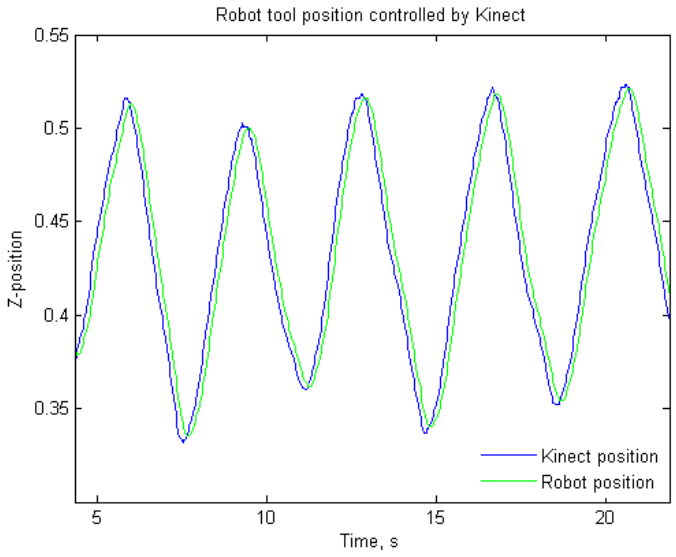
signals with high amplitudes at low frequencies, but this will decrease the response at higher frequencies. So it is clear that the gain has to be selected based on the desired amplitudes and frequencies the system is supposed to track.

7.2 Demonstration of how the Kinect directly controls the robot's position

Figure 7.4 on the next page shows how the robot tool positon is controlled with the Kinect's depth measurements. The graph in figure 7.4b shows the position commands sent from the Kinect process and the robot's position. This test indicate a delay of 0.1167*s* between a position is found by the Kinect and the robot is at that position. The delay is calculated as the average delay from when the Kinect data crosses the 0.45 line and the robot position crosses it.



(a) Two shots from the demonstration of how the robot follows a surface tracked by the Kinect



(b) Position command sent from Kinect, compared to real robot position

Figure 7.4: Demonstration of robot tool position controlled by the Kinect.

Part IV

Discussion and conclusion

Chapter 8

Discussion

8.1 Discussion of the test results

8.1.1 Kinect sensor

A lot of different tests performed on the Kinect have been presented in chapters 4 and 5. A common result in all of them has been that the Kinect has had the best precision and accuracy on the lower limits of its operative range, and that the precision decreases as the distance between the Kinect and the objects in sight increases. Considering single pixels in the Kinect's depth image, it is clear that depth resolution can be a problem. But it has been shown that by averaging the depth values from a small neighborhood of pixels, the depth resolution will increase. This comes at the cost of reduced quality around sharp edges, but the measurements around sharp edges are already very corrupted by noise, so this is in many situations a price worth paying.

Since the Kinect measurements are best at close range it is clear that measurements should be taken as close as possible, but it is then important to remember that the Kinect's field of view decreases when the distance decreases. So to get a large enough view of the patient the Kinect should be placed at least $75cm$ away. This will give a field of view of $\approx 80x60cm$, which should be enough to cover a persons upper body.

The Kinect's bandwidth was not completely tested since the robot could not follow a signal faster than $3Hz$. From the Kinect's frame rate and the sampling theorem we know that in theory the Kinect can track periodic signals of up to $15Hz$. As shown in figure 4.6 in section 4.8.1 the theoretical response in the Kinect's sampling slowly decreased and between 13–$14Hz$ the gain dropped below $-3dB$, which is the normal cut-off level for defining bandwidth. When the Kinect was used to measure the periodic movement of the robot tool, it became clear that the gain did not decrease but stayed on a level between $0dB$ and $1dB$, with two peaks of $\approx 2dB$. This high gain was caused by the Kinect's low resolution compared to the amplitude of the robot movements, and could probably have been avoided if the position commands were taken from multiple pixels and not just one. The average time delay between the robot's position and the position measured by the Kinect was found to be $\approx 0.04s$.

8.1.2 Kinect – robot system

The low bandwidth and the high phase delay in the system was a bit surprising. With a maximum tool speed of $1m/s$ it was expected that the system would be able to track higher frequencies and amplitudes than it did. Three possible reasons for this result are:

1. The robot cannot operate any faster
2. The position updates are to slow
3. The control algorithm isn't good enough

Alternative 1 is not the case. When the interface between the Unix computer and the robot was set up, a test of the bandwidth was performed[1]. That test used a different control method than here, but the motion was controlled by a sine wave. The exact results from this test is not known, but the results were much better than here, with almost no delay. So it is clear that the system can perform better with a different control method.

Case 2 is more presumable. Since the velocity commands to the robot are based on the error between the current position and the desired position, the response will be slower if the velocity commands are updated more often than the position commands. This is because there will be an internal time delay in the whole system. For each new position command from the Kinect, the computer sends approximately four velocity commands to the robot. For each command, the error between current position and desired position is slightly decreased. This causes the velocity to decrease as well for each time step. Then when a new position command is ready, the error increases and so does the velocity, so the velocity will decrease and increase between new position commands. If the position commands had been updated with the same interval as the velocity, then the error between current and desired position would be more stable, producing a more stable velocity. So the response would probably improve if the position commands were updated more often.

The final alternative is that the implemented controller isn't good enough. As mentioned before, the system uses a proportional controller. This is among the simplest of controllers. By implementing a more advanced controller, for instance a PID (Proportional-Integral-Derivative) controller [51, p.224], the total response in the system would probably perform better.

8.2 The use of Kinect for motion compensation

The total delay in the system was never directly tested, but tested in two steps. The delay between the robot's movement and the Kinect's measurements was found (section 5.8), and then the delay between a position control signal and the robot's position was found (section 7.1). An indication of the total delay can be found by combining these two delays. In addition the delay from the demonstration of how the robot was directly controlled by data from the Kinect (section 7.2) can be used to confirm this. The delay between robot position and Kinect measurement found in section 5.8 was $0.0399s \approx 0.04s$. With the sine wave position control, the time between a position command was given to the robot and until the robot got there was $0.0952s \approx 0.1s$. This gives a total delay of $\approx 0.1s + 0.04s = 0.14s$. When the robot was controlled with position commands directly from the robot, the delay was found to be $0.1167s \approx 0.12s$. This is only the delay between a new frame from the Kinect and the robot's position, so an extra $1/30s = 0.033s$ must be added for the

[1] This was not part of this thesis, but the Ph.D study.

Kinect's frame rate. This brings the total delay up to $0.12s + 0.3s = 0.15s$. According to [43] the maximum allowed latency in tele-surgery is $0.2s$ so in both cases the delay is within this limit. And it is assumed that there is a bit more slack in the latency requirements for ultrasound diagnostics than it is for tele-surgery, so the time delay in the system will not be a problem. The fact that delay is $< 0.2s$ just opens up a wider range of applications for the system, for instance respiration tracking in radio-surgery [44].

In this system the wanted application for the Kinect was to track patient movement and then send position commands to the robot, so it could keep a fixed relation between the robot tool and the patient. The main source of motion in the patient will most likely come from respiratory motion. The normal respiratory rate of a person is approximately 12 breaths pr. minute [23, p.477] and can occasionally rise to 40 or 50 breaths pr. minute. So on average a person breathes with a frequency of $0.2Hz$, but the breathing can also occur at $\approx 0.83Hz$. From the numbers in table 7.1 on page 82 we can see that the Kinect–robot system can track and follow the breathing at $0.2Hz$ with a gain of approximately 99%. For the highest breath rate, the system can track the motion with a gain between 85% and 90%. At this rate the delay will be around 0.1s and 0.13s. When it comes to the amplitude of respiratory motion, it is harder to find exact numbers. The movement will vary, depending on where on the body the measurements are taken. For instance, the upper part of the chest will normally only make small movements, while the abdominal will move with greater amplitude. The movement can vary from a few millimeters [20], to several centimeters [29]. So from this is is clear that at the normal breathing frequency of $0.2Hz$, the major limitation in the system is the Kinect's resolution, as it is not certain that the Kinect will detect a $2mm$ change of position.

In the system tests, the Kinect has been used to give direct position commands to control the robot's tool position. This can be useful for directing the robot to the correct parts of the patient's body, but when the ultrasound probe reaches the patient, other control methods should take over. The Kinect will not see the impact point between the patient and ultrasound probe, and even though hole filling algorithms might be used to estimate the surface below the probe, the estimation will be inaccurate since the surface isn't solid and the probe will deform the patient's skin. Therefor the best solution would be to let the force sensor, and maybe the ultrasound image, control the final positioning of the ultrasound probe. And thus the data from the force sensor is a better source to compensate for patient movement at the probe position. But the Kinect can still be used to track the motion of the patient where there is no force feedback. With only force feedback from the robot's tool, there is the chance that other parts of the robot will collide with the patient. Thus the data from the Kinect can be used to track the rest of the patient's body, this can be used to check if any part of the robot is on collision course with the patient. This surface map of the patient can be updated at $30Hz$ while the rest of the controller runs at $125Hz$. For every position update, the control program can use the last surface scan from the Kinect to check if any part of the robot is to near the patient, and in that case calculate an alternative configuration for the joints.

8.3 Position commands via Ethernet connection

The tests on the Kinect–robot interface were performed by using the real-time interface between the Unix computer and the robot. One alternative way to send instructions to the robot is via the Ethernet connection. This control approach was tested by Mektron. The UR has an internal script that can be used to send direct position commands to the robot with the Ethernet interface. So in their test setup, new position commands from the Kinect was sent to the robot every time a new frame was ready from the Kinect. When a new position was sent to the robot it stopped completely before it started moving towards the next position. Since the position was updated with $30Hz$ the result was that the robot just vibrated around the same position and did not move much. As a solution to this they reduced the rate of position updates from the Kinect. Then the robot had time to move before a new command arrived, but this caused a major time delay and was thus not very useful.

Without explaining how, Universal Robots reported that the University of Munich have achieved good results with position commands via this interface[2]. It is not known how it was done, but they have achieved position updates at 100Hz. At this frame rate they clearly haven't used the Kinect to give direction commands, but this indicates that it should be possible to achieve better performance with the Ethernet connection than Mektron's initial test did.

8.4 One or two Kinects?

We have seen that by using one extra Kinect in the depth measurements, the precision is reduced because of interference between the Kinects. Even with this loss of precision there are still some major advantages with multiple Kinects.

- Larger field of view

  By using two Kinects it will be possible to monitor the entire upper body of the patient. This will increase the operative space for the robot, as it then can work on the entire body and just not on one side.

- Shadow reduction

  When the robot comes close to the patient, it will shadow the IR light emitted from the Kinect. This will reduce the visible surface of the patient, and the Kinect will not be able to model the surface behind the robot. With two Kinect, the area that is occluded by the robot for one of the Kinects may be visible to the other. This way a full model of the patient can be kept, even when the robot has contact with the patient. The only "invisible" point will be the contact

[2]This information was given via Mektron by mail on 7.Mai 2012

point between the ultrasound probe and the patient. The interference between the Kinects may even be reduced because of the robot.

Solutions to interference problem

There are several ways to reduce the interference between multiple Kinects, [45, 32, 31]. In [45] a time division multiplexing approach, where the IR emitter on each Kinect is blocked in turn so that the IR patterns don't interfere, are proposed. The results are very good, but the frame rate of good depth images are reduces with the number of Kinects. So this may be a good solution for multi-view surface reconstruction, but it cannot be used in a real-time position control system.

[31] reduces the interference by continuously moving the Kinects. The idea is that when the Kinects move, the IR pattern projected by the other Kinects will be blurred while the Kinect still can see its own pattern. This method reduced the interference, but the movement of the Kinects introduces the need for constant image registration as well. Thus the system is per now not suited for real-time control.

Finally [32] presents a method that doesn't prevent the interference, but uses a hole filling algorithm to fill missing depth caused by the interference, and median filtering to reduce the noise. They also present a method for real-time merging of the data from multiple Kinects. Their system can successfully keep the Kinect's frame rate of 30 fps. So it seem like a hole filling procedure is the best solution for this system. The same hole filling algorithm could then be used to estimate the surface of the patient in areas where the robot blocks the IR light.

8.5 Alternative devices

Kinect for Windows

In January 2012 Microsoft released a new version of the Kinect, Kinect for Windows[3]. This new device is made with the single purpose of making computer applications with Kinect. Microsoft claims that this is a new and improved version of the Xbox Kinect, but this is yet to be tested. The technical details reported by Microsoft for Kinect for Windows are the same as for the Xbox Kinect, so the improvements must lie in the firmware. One of the new features in Kinect for Windows is the "Near mode" where one can access depth readings from the Kinect for distances down to $40cm$. For Kinect developers using the Microsoft SDK this is probably a new and improved feature, but for user with OpenNI software depth readings has already been available at $50cm$ distance so this is not a big improvement. And as long as the angular field of view is the same as in the Xbox Kinect, the Kinect for Windows will probably not see enough of the patient if used in near mode at $40cm$. At the moment it is not possible to use the Kinect for Windows sensor with OpenNI, it will only run with the Kinect

[3] http://www.microsoft.com/en-us/kinectforwindows/

for Windows SDK. So the Kinect for Windows will have to be run from a different computer than the robot, and the possibility of real-control will be removed.

Asus Xtion

Asus has made two devices similar to the Kinect, Xtion Pro and Xtion Pro Live. Xtion Pro gives you only the depth readings, while Xtion Pro Live is similar to the Kinect with depth sensor, RGB camera and two microphones. The Xtion and the Kinect is built around the same PrimeSense technology, so the technical details are very similar, but there are some features that can make Xtion advantageous compared to the Kinect in robotic applications.

- Higher frame rate

  With the Xtion you can choose to decrease the resolution from $640x480$ to $320x240$ and increase the frame rate from 30fps to 60fps. We have seen that the Kinect's frame rate might be a problem, so this increase of frame rate might be useful.

- Physical size

  The Xtion cameras are both smaller in volume and weighs less than the Kinect. This can make them easier to position, or easier to integrate in mobile robots where weight may be an issue.

- Power supply

  In contrast to the Kinect that needs an extra power cable to work with a computer, the Xtion gets all the power it need from the computer's USB ports. This is not a big deal if your application is placed at a fixed location, but if you want to move it around it is much easier if you just need the USB connection.

- Wireless

  The Xtion can be connected to Asus Wavi, a wireless transmitter with a range of 25m. Again a feature that can be very useful for mobile robots.

8.6 Suggestions for future work

It is yet to be tested how the Kinect's capabilities are affected with the robot arm partially blocking its view. It should also be made an implementation that uses two Kinects to increase the system's field of view and reduces the shadow from the robot. An extra Kinect is easily added to the existing interface by adding a new *pcl::OpenNIGrabber* pointer and callback function to the code.

An alternative control algorithm should be implemented that uses the data from the Kinect to generate position constraints to the robot joints. This was explained at the end of section 8.2.

A test using Xtion Pro should be performed to see if the the increased frame rate can improve the results in direct position control. Xtion can be used with OpenNI, so the existing interface can be used for this. Kinect for Windows is another test alternative, but this has to be used from a Windows computer and interfaced via the Ethernet connection.

Chapter 9

Conclusion

9.1 Conclusion

In this thesis it has been conducted a set of tests to investigate the Kinect's performance when it comes to precision, accuracy and bandwidth. The Kinect has shown to be most accurate and precise on short distances, so measurements should be taken as close as possible. But it is important to remember that when the distance decreases, so does the Kinect's field of view. The precision and resolution decreases rapidly with the distance, so depth measurements should be kept at a distance below $1.5m$, this will give a depth accuracy $> 90\%$. Pixel by pixel there is some noise, so the precision will be better by generating a smooth surface model instead of using the direct depth values from the Kinect. The Kinects upper bandwidth boundary was not found, but the delay between a movement was made and the position was reported by the Kinect was found to be $\approx 0.04s$

At low frequencies ($< 1Hz$) the Kinect–robot system could follow periodic position commands with an peak-to-peak amplitude of up to $10cm$, so there should be no problems with tracking normal respiratory motions. The system's phase delay was higher than expected, but the total time delay was found to be $< 0.2s$, so the delay is within the limits of what is allowed in tele-surgery.

When in comes to motion compensation, the Kinect can not be used at the impact point between the patient and the ultrasound probe, this comes from the fact that the Kinect will not be able to see the surface, and creating an accurate model based on the surrounding surface will be difficult, since the pressure applied by the robot on the patient skin will deform the skin surface. Therefor a force sensor will be a better source of feedback to compensate for the movement. But the Kinect can still be used to make sure that the other parts of the robot keeps a safe distance from the patient. The visual data from the Kinect can best be used to maneuver the robot to the correct part of the body, while the force sensor has to be used for the final positioning.

Bibliography

[1] http://www.ros.org/wiki/kinect_calibration/technical.

[2] http://en.wikipedia.org/wiki/History_of_video_games.

[3] www.wikipedia.org/wiki/Cathode_ray_tube_amusement_device.

[4] http://classicgames.about.com/od/history/tp/ClassicVGTimelinePart1.htm.

[5] http://classicgames.about.com/od/classicvideogames101/p/MagnavoxOdyssey.htm.

[6] http://en.wikipedia.org/wiki/History_of_video_game_consoles_%28second_generation%29.

[7] http://en.wikipedia.org/wiki/History_of_video_game_consoles_%28fifth_generation%29#Transition_to_3D.

[8] http://news.bbc.co.uk/2/hi/in_depth/sci_tech/2001/artificial_intelligence/1531432.stm.

[9] http://www.moah.org/exhibits/archives/robotman/history/history.html.

[10] http://www.pbs.org/tesla/ll/ll_robots.html.

[11] http://www.thocp.net/reference/robotics/robotics2.htm#26.

[12] http://www.computerhistory.org/revolution/artificial-intelligence-robotics/13/289.

[13] http://en.wikipedia.org/wiki/Telerobotics.

[14] http://en.wikipedia.org/wiki/Autonomous_robot.

[15] http://communities.canada.com/ottawacitizen/blogs/defencewatch/archive/2011/03/17/video-game-technology-used-to-counter-ieds.aspx.

[16] http://www.physorg.com/news/2010-12-air-playstation-3s-supercomputer.html.

[17] http://www.ifixit.com/Teardown/Microsoft-Kinect-Teardown/4066/.

[18] http://www.mathworks.com/matlabcentral/fileexchange/34129.

[19] Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14:239–256, February 1992.

[20] Sonja Dieterich, Jonathan Tang, James Rodgers, and Kevin Cleary. Skin respiratory motion tracking for stereotactic radiosurgery using the cyberknife. *International Congress Series*, 1256(0):130 – 136, 2003. CARS 2003. Computer Assisted Radiology and Surgery. Proceedings of the 17th International Congress and Exhibition.

[21] J. Faust, C. Simon, and W.D. Smart. A video game-based mobile robot simulation environment. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 3749 –3754, oct. 2006.

[22] L. Gallo, A.P. Placitelli, and M. Ciampi. Controller-free exploration of medical image data: Experiencing the kinect. In *Computer-Based Medical Systems (CBMS), 2011 24th International Symposium on*, pages 1 –6, june 2011.

[23] Guyton and Hall. *Textbook of Medical Physiology*. Elsevier Saunders, 11th edition, 2006.

[24] M. Hebert. Active and passive range sensing for robotics. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 1, pages 102 –110 vol.1, 2000.

[25] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy. Visual odometry and mapping for autonomous flight using an rgb-d camera. In *Int. Symposium on Robotics Research (ISRR), (Flagstaff, Arizona,USA)*, Aug 2011.

[26] Timo Kahlmann. *Range Imaging Metrology: Investigation, Calibration and Development*. PhD thesis, University of Hannover, 2007.

[27] K. Khoshelham. Accuracy analysis of kinect depth data. In *ISPRS Workshop Laser Scanning 2011*, Calgary, Canada, August 29-31 2011.

[28] Sung-Yeol Kim, Eun-Kyung Lee, and Yo-Sung Ho. Generation of roi enhanced depth maps using stereoscopic cameras and a depth camera. *Broadcasting, IEEE Transactions on*, 54(4):732 –740, dec. 2008.

[29] Vijay R Kini, Subrahmanya S Vedam, Paul J Keall, Sumukh Patil, Clayton Chen, and Radhe Mohan. Patient training in respiratory-gated radiotherapy. *Medical Dosimetry*, 28(1):7 – 11, 2003.

[30] Michael Krainin, Peter Henry, Xiaofeng Ren, and Dieter Fox. Manipulator and object tracking for in-hand 3d object modeling. *Int. J. Rob. Res.*, 30(11):1311–1327, sep 2011.

[31] A. Maimone and H. Fuchs. Reducing interference between multiple structured light depth sensors using motion. In *Virtual Reality (VR), 2012 IEEE*, pages 51 –54, march 2012.

[32] Andrew Maimone and Henry Fuchs. Encumbrance-free telepresence system with real-time 3d capture and display using commodity depth cameras. *Mixed and Augmented Reality, IEEE / ACM International Symposium on*, 0:137–146, 2011.

[33] A. Makadia, A.I. Patterson, and K. Daniilidis. Fully automatic registration of 3d point clouds. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 1297 – 1304, june 2006.

[34] Microsoft. Kinect™for windows®sdk beta, programming guide, getting started with the kinect for windows sdk beta from microsoft research. Beta 1 Draft version 1.1 - July 22, 2011.

[35] M. Mitsuishi, S. Warisawa, T. Tsuda, T. Higuchi, N. Koizumi, H. Hashizume, and K. Fujiwara. Remote ultrasound diagnostic system. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 2, pages 1567 – 1574 vol.2, 2001.

[36] Brent Mittelstadt, Howard Paul, Peter Kazanzides, Joel Zuhars, Bill Williamson, Robert Pettitt, Phillip Cain, David Kloth, Luke Rose, and Bela Musits. Development of a surgical robot for cementless total hip replacement. *Robotica*, 11(06):553–560, 1993.

[37] Shimon Y. Nof. *Handbook of Industrial Robotics*. Wiley, 2nd edition, 1999.

[38] PrimeSense. Primesense-3d-sensor-data-sheet. http://www.primesense.com/en/press-room/resources/file/4-primesense-3d-sensor-data-sheet.

[39] Guinness World Records. http://community.guinnessworldrecords.com/_Kinect-Confirmed-As-Fastest-Selling-Consumer-Electronics-Device/blog/3376939/7691.html.

[40] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[41] Fredrik Ryden, Howard Chizeck, Sina Nia Kosari, Hawkeye King, and Blake Hannaford. Using kinect and a haptic interface for implementation of real-time virtual fixtures. In *Proceedings of the 2nd Workshop on RGB-D: Advanced Reasoning with Depth Cameras (in conjunction with RSS 2011)*, Los Angeles, USA, June 27 2011.

[42] S. Salcudean, G. Bell, S. Bachmann, W. Zhu, P. Abolmaesumi, and P. Lawrence. Robot-assisted diagnostic ultrasound - design and feasibility experiments. In Chris Taylor and Alain Colchester, editors, *Medical Image Computing and Computer-Assisted Intervention - MICCAI 99*, volume 1679 of *Lecture Notes in Computer Science*, pages 1062–1071. Springer Berlin / Heidelberg, 1999.

[43] Richard Satava. Surgical robotics: The early chronicles: A personal historical perspective. *Surgical Laparoscopy, Endoscopy & Percutaneous Techniques*, 12:6–16, 2002.

[44] Joel Schaerer, Aurora Fassi, Marco Riboldi, Pietro Cerveri, Guido Baroni, and David Sarrut. Multi-dimensional respiratory motion tracking from markerless optical surface imaging based on deformable mesh registration. *Physics in Medicine and Biology*, 57(2):357, 2012.

[45] Yannic Schröder, Alexander Scholz, Kai Berger, Kai Ruhl, Stefan Guthe, and Marcus Magnor. Multiple kinect studies. Technical Report 09-15, ICG, oct 2011.

[46] J. Smisek, M. Jancosek, and T. Pajdla. 3d with kinect. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 1154 –1160, nov. 2011.

[47] Spong, Hutchinson, and Vidyasagar. *Robot modeling and control*. Wiley, 1st edition, 2006.

[48] J. Stowers, M. Hayes, and A. Bainbridge-Smith. Altitude control of a quadrotor helicopter using depth map from microsoft kinect sensor. In *Mechatronics (ICM), 2011 IEEE International Conference on*, pages 358 –362, april 2011.

[49] Todor Stoyanov, Athanasia Louloudi, Henrik Andreasson, and Achin J. Lilienthal. Comparative evaluation of range sensor accuracy in indoor environments. In *5th European Conference on Mobile Robots, ECMR 2011*, Orebro, Sweden, September 7-9 2011.

[50] Jocelyne Troccaz. Computer and robot-assisted medical intervention. In Shimon Y. Nof, editor, *Springer Handbook of Automation*, pages 1451–1466. Springer Berlin Heidelberg, 2009.

[51] Arne Tyssø. *Automatiseringsteknikk. Modellering, Analyse og syntese av reguleringssystemer.* NKI forlaget, 2nd edition, 1987.

[52] Vanderpool, Friis, Smith, and Harms. Prevalence of carpal tunnel syndrome and other work-related musculoskeletal problems in cardiac sonographers. *Journal of occupational medicine*, 35:604–610, 1993.

[53] Adriana Vilchis Gonzales, Philippe Cinquin, Jocelyne Troccaz, Agnes Guerraz, Bernard Hennion, Franck Pellissier, Pierre Thorel, Fabien Courreges, Alain Gourdon, Gerard Poisson, Pierre Vieyres, Pierre Caron, Olivier Merigeaux, Loic Urbain, Cedric Daimo, Stephane Lavallee, Philippe Arbeille, Marc Althuser, Jean-Marc Ayoubi, Bertrand Tondu, and Serge Ippolito. Ter: A system for robotic teleechography. In Wiro Niessen and Max Viergever, editors, *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2001*, volume 2208 of *Lecture Notes in Computer Science*, pages 326–334. Springer Berlin / Heidelberg, 2001.

[54] Mark J. P. Wolf. *The Video Game Explosion, A history from pong to playstation and beyond*. Greenwood Press, London, 2008.

Appendix A

Initzialization time

After the Kinect start taking depth measurements it seem like it need some time to calibrate before the depth readings stabilizes. This can be seen in figure A.1 on the next page. This figure shows the average and standard deviation in measured distance between the Kinect and a flat wall for 6 different distances for a 100 frames at each distance. We can see that after approximately 40 frames there is a sudden change in the average distance and standard deviation.

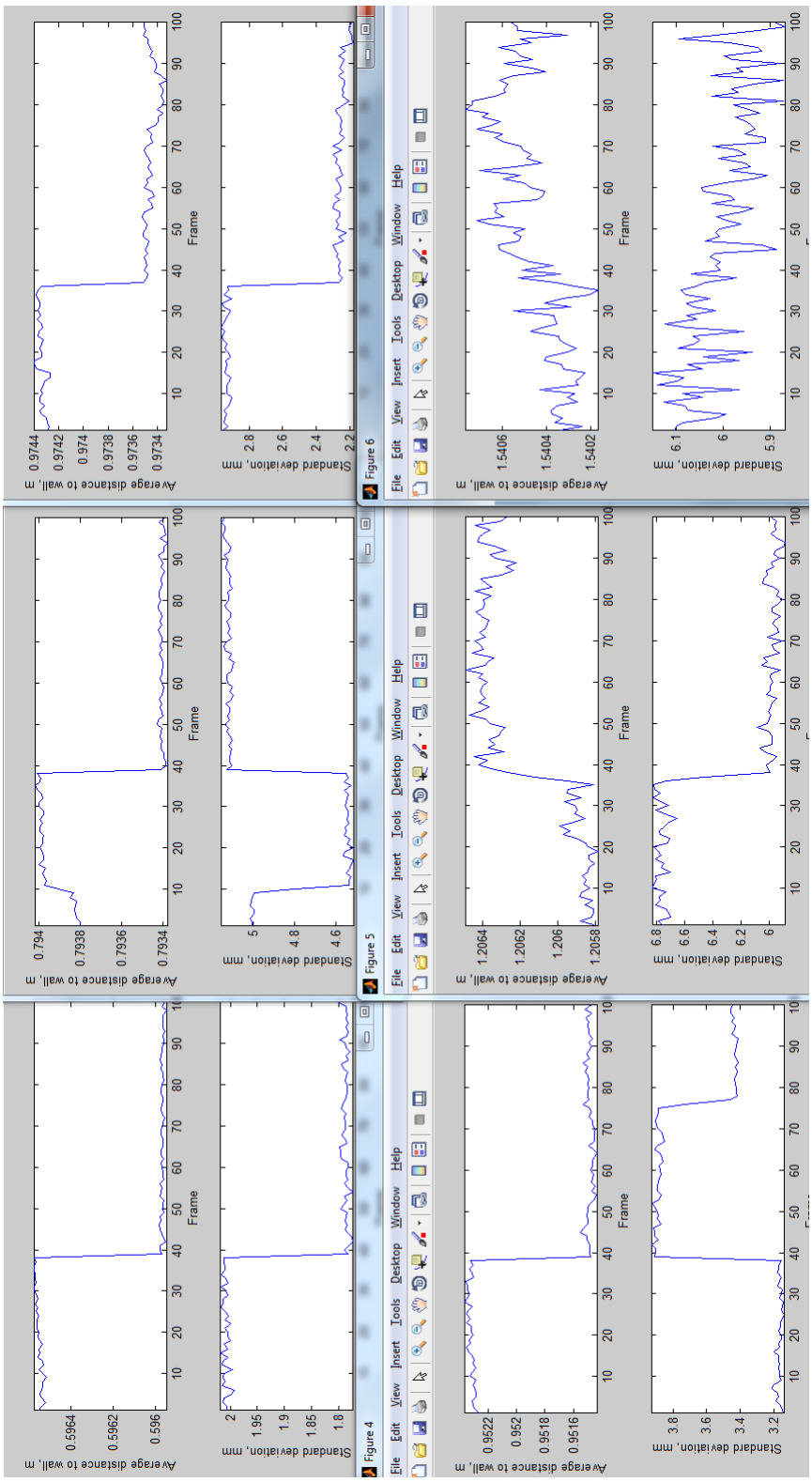


Figure A.1: Drop in measured depth with buffered frames

Appendix B

Kinect–robot Interface

This is the code used to interface the Kinect with the robot system. In the current state, parameters are set as they were in the final demonstration of how the Kinect could directly be used to control the robot.

```
#include <pcl/io/openni_grabber.h>
#include <pcl/point_types.h>

#include <math.h>
#include <iostream>

using namespace std;

typedef pcl::PointXYZ PointT;
void findNextPosition(pcl::PointCloud< PointT >::ConstPtr cloud);
void kinectGetCenterPointDistance(pcl::PointCloud< PointT >::
    ConstPtr cloud);
void kinectGetLastPosition(double *pos);

pcl::OpenNIGrabber *kinectInterface; //!< A pointer to the kinect
    interface
pcl::PointCloud<PointT>::ConstPtr kinectPointCloud; //!< The last
    point cloud received from the Kinect
Eigen::Vector4f kinectLastPosition, kinectNewPosition; //!< Stores
     the centerpoint of the tracked plane
double kinectRotation[9]; //!< Wanted orientation on the end
    effector
boost::mutex kinectCloudMutex, kinectPointMutex;


Eigen::Matrix4f kinectTransMat; //!< The translation matrix to
    rotate from kinect coordinate to robot base coordinate

//! Callback function for the Kinect depth data
/*!
* Receives a new point cloud from the Kinect and calls
    kinectFindPlane
* Also prints statistics about the Kinect frame rate
*/
void cloud_cb (const pcl::PointCloud<PointT>::ConstPtr & cloud) {
  if (kinectCloudMutex.try_lock()) {
    kinectPointCloud = cloud;
//    kinectNewCloud = true;
```

```
    findNextPosition(cloud);
33    kinectCloudMutex.unlock ();
  }
35}

37//! Finds the next position the robot should move to
/*!
39 * Method to find the next position the robot should move to. This
       method is
 * up to the user to implement. The found position should be found
       in the
41 * Kinect's coordinate system and transformed to the robot's
     coordinate system
 * with the function transformPoint();
43 */
void findNextPosition(pcl::PointCloud< PointT >::ConstPtr cloud) {
45  kinectGetCenterPointDistance(cloud);
}
47
//! Retreive the last position from the Kinect
49/*!
 * Stores the last found position from the Kinect in the pos array
       .
51 * If the position array is busy, the last position is returned.
* @param double pos[] - Array to store the (x,y,z) coordinates in
53*/
void kinectGetLastPosition(double *pos) {
55  if (kinectPointMutex.try_lock()) {
    kinectLastPosition(0) = kinectNewPosition(0);
57    kinectLastPosition(1) = kinectNewPosition(1);
    kinectLastPosition(2) = kinectNewPosition(2);
59    kinectPointMutex.unlock();
  }
61  pos[0] = kinectLastPosition(0);
  pos[1] = kinectLastPosition(1);
63  pos[2] = kinectLastPosition(2);
}
65
//! Transforms the point from Kinect coordinate system to robot
     base coordinate system
67/*!
* Stores the transformed point in kinectPlaneCenterPoint
69* The transformation matrix used is defind in initializeKinect().
*
71* @param Eigen::vector4f - the point to transform from kinect to
     robot
*/
73void transformPoint(Eigen::Vector4f centerPoint) {
  centerPoint = kinectTransMat*centerPoint;
75  if(kinectPointMutex.try_lock()) {
    kinectNewPosition = centerPoint;
77    kinectPointMutex.unlock();
  }
79}

81//! Test method to only control one of the robot axis
/*!
83* Finds the distance to the centerpoint and transforms it to the
     robot coordinate
```

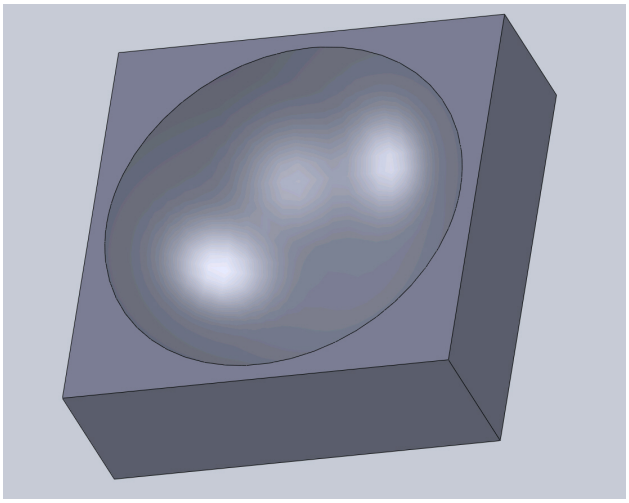
```
*/
void kinectGetCenterPointDistance(pcl::PointCloud< PointT >::
    ConstPtr cloud){
  Eigen::Vector4f centerPoint;
  centerPoint(0) = 0.0;
  centerPoint(1) = 0.0;
  // Check for missing depth value
  if (cloud->points[320*481].z == cloud->points[320*481].z) {
    centerPoint(2) = cloud->points[320*481].z;
    transformPoint(centerPoint);
  }
}

//! Initializes the Kinect interface and sets all global variables
/*!
* Initializes a new OpenNIGrabber and all global variables needed
    for the Kinect control.
*/
void initializeKinect() {
  printf("Starting kinectInterface\n");
  kinectInterface = new pcl::OpenNIGrabber();
  boost::function<void (const pcl::PointCloud<pcl::PointXYZ>::
      ConstPtr&)> f1 =
    boost::bind(&cloud_cb, _1);
  boost::signals2::connection c1 = kinectInterface->
      registerCallback(f1);
  kinectTransMat << 1.0, 0.0, 0.0, 0.00,
          0.0, 1.0, 0.0, 0.00,
          0.0, 0.0, 1.0, 0.00,
          0.0, 0.0, 0.0, 1.00;

  // Set starting position commands to default positions
  kinectNewPosition(0) = -0.5;
  kinectNewPosition(1) = -0.11;
  kinectNewPosition(2) = 0.45;
  kinectLastPosition(0) = -0.5;
  kinectLastPosition(1) = -0.11;
  kinectLastPosition(2) = 0.45;

//  kinectNewCloud = false;
  kinectRotation[0] = 0; //Sets the robot tool to pointing down
  kinectRotation[1] = 1;
  kinectRotation[2] = 0;
  kinectRotation[3] = 1;
  kinectRotation[4] = 0;
  kinectRotation[5] = 0;
  kinectRotation[6] = 0;
  kinectRotation[7] = 0;
  kinectRotation[8] = -1;

  kinectInterface->start();
}

//!
/*!
* Stops the Kinect process
*/
void stopKinect() {
  printf("Closing down Kinect\n");
```

```
139     kinectInterface->stop();
}
```

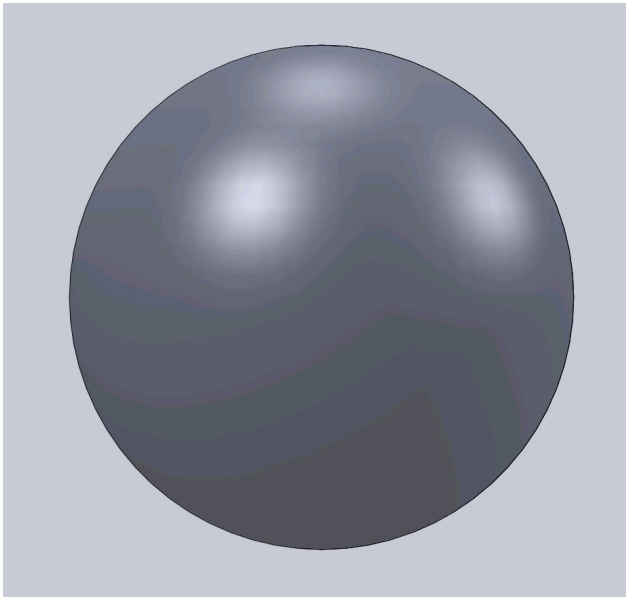
Listing B.1: kinectRobotInterface.h

Appendix C

Testmodels



(a) Solidworks model of half sphere

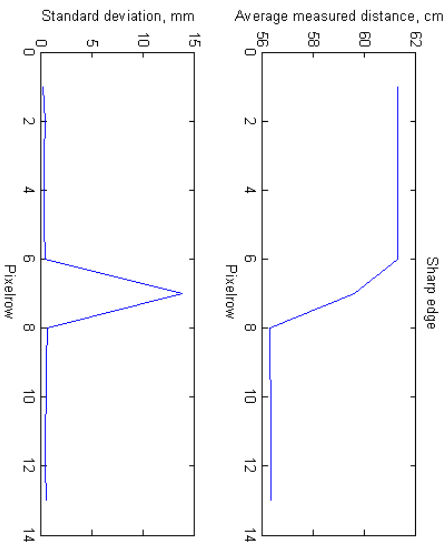


(b) Solidworks model of sphere

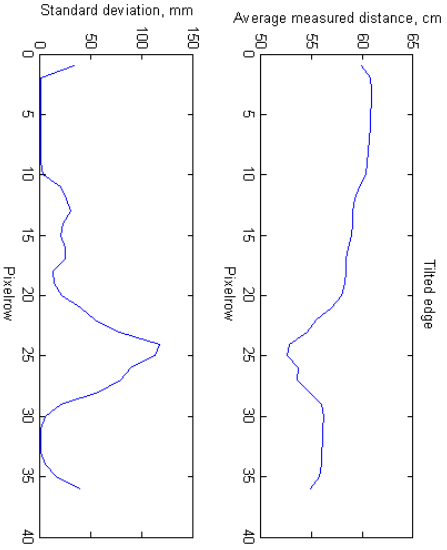
Figure C.1: Solidworks models

Appendix D

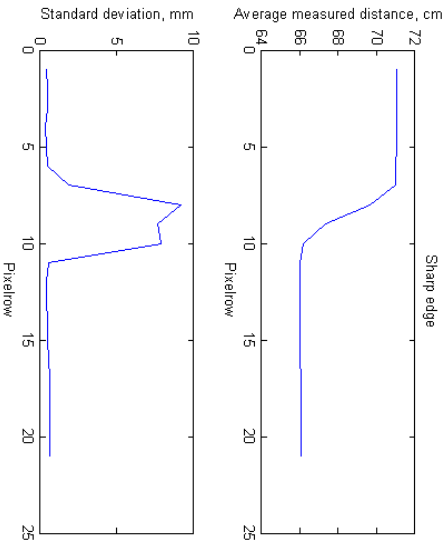
Edgegraphs



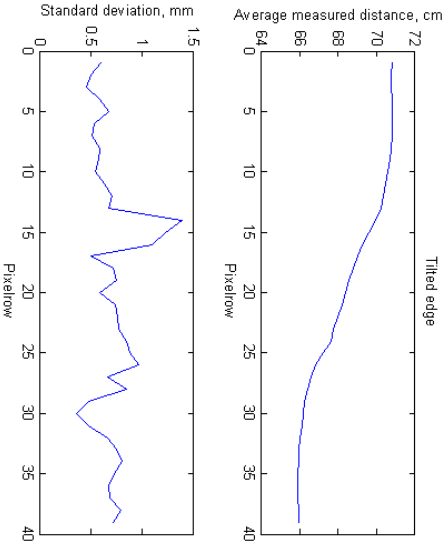
(a) Average depth along x-axis



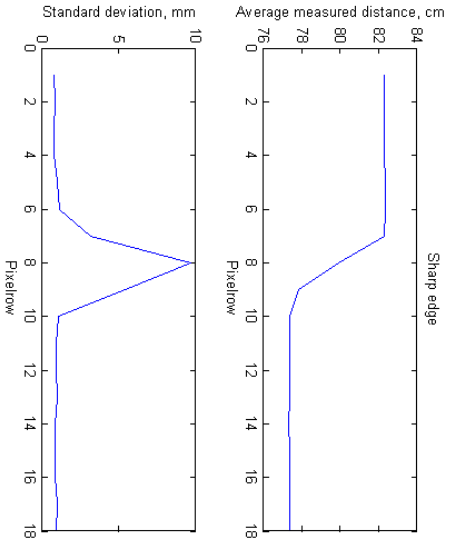
(b) Average depth along x-axis



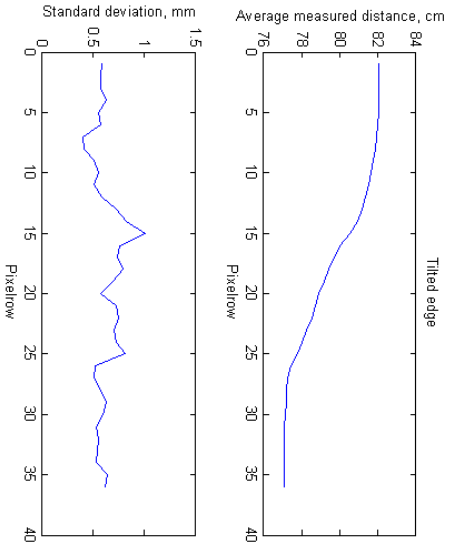
(a) Average depth along x-axis



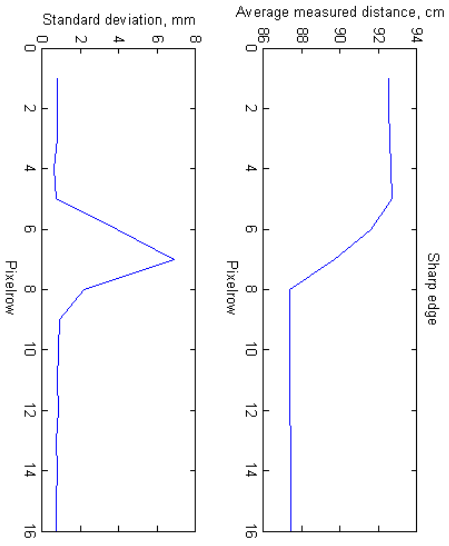
(b) Average depth along x-axis



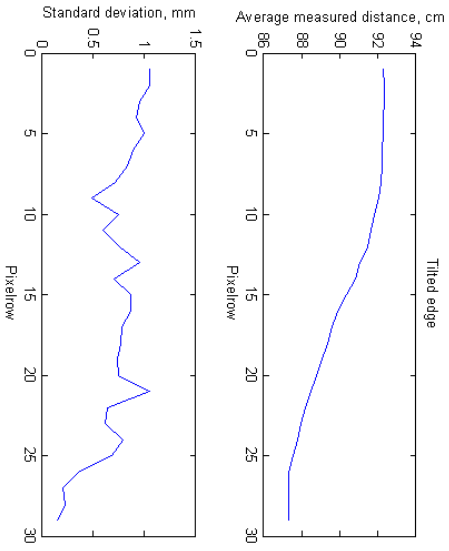
(a) Average depth along x-axis



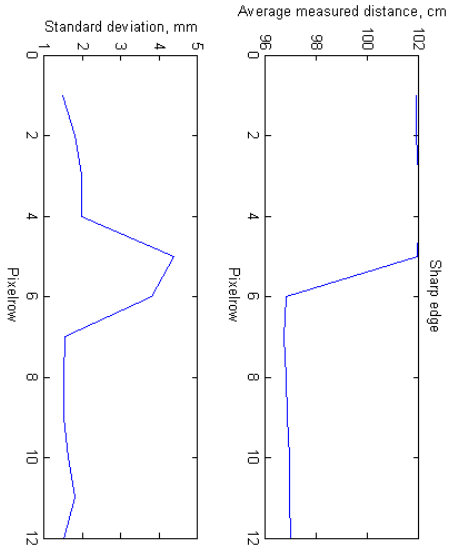
(b) Average depth along x-axis



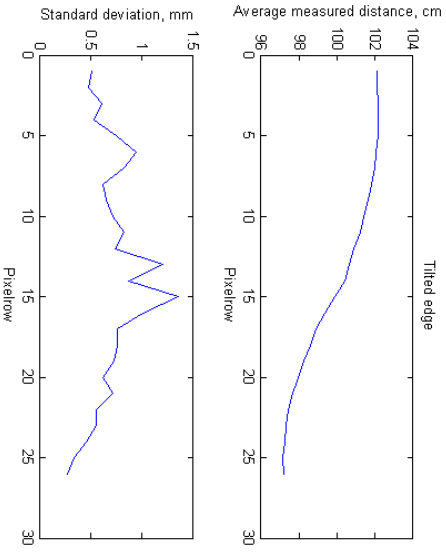
(a) Average depth along x-axis



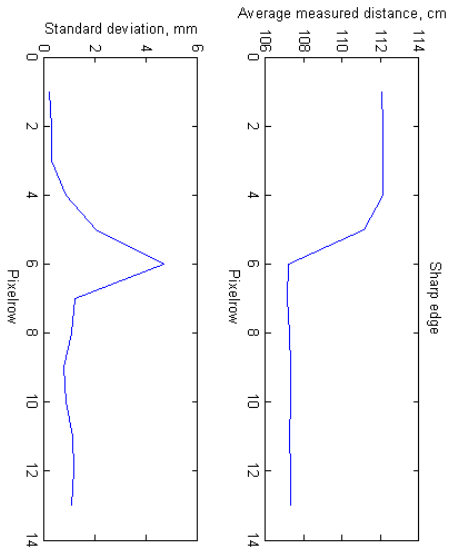
(b) Average depth along x-axis



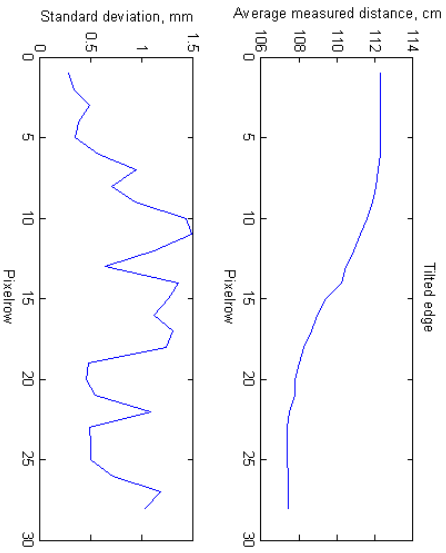
(a) Average depth along x-axis



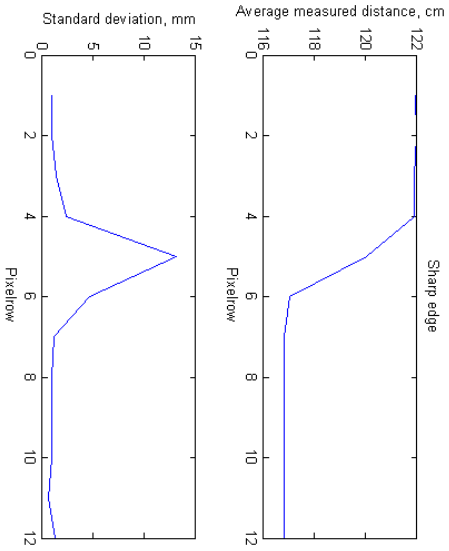
(b) Average depth along x-axis



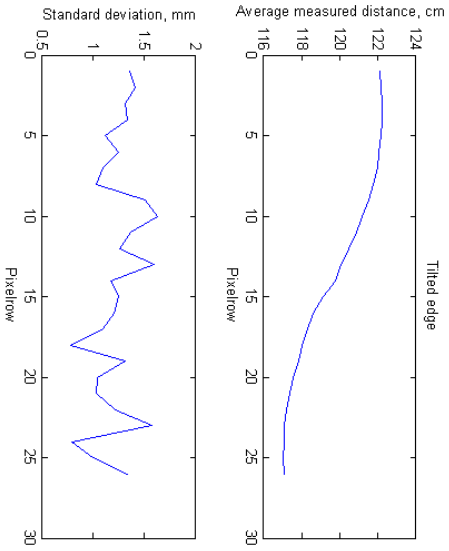
(a) Average depth along x-axis



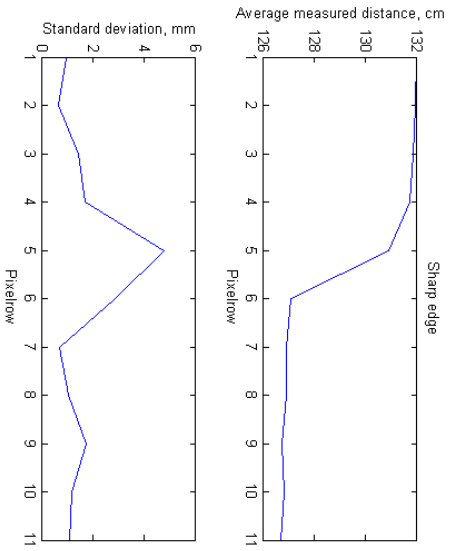
(b) Average depth along x-axis



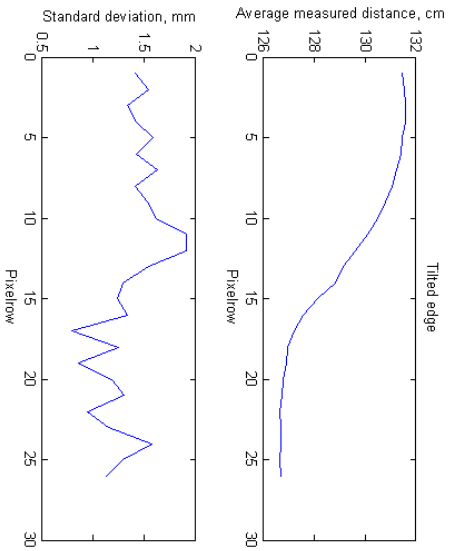
(a) Average depth along x-axis



(b) Average depth along x-axis



(a) Average depth along x-axis



(b) Average depth along x-axis

Appendix E

Kinect Tracking

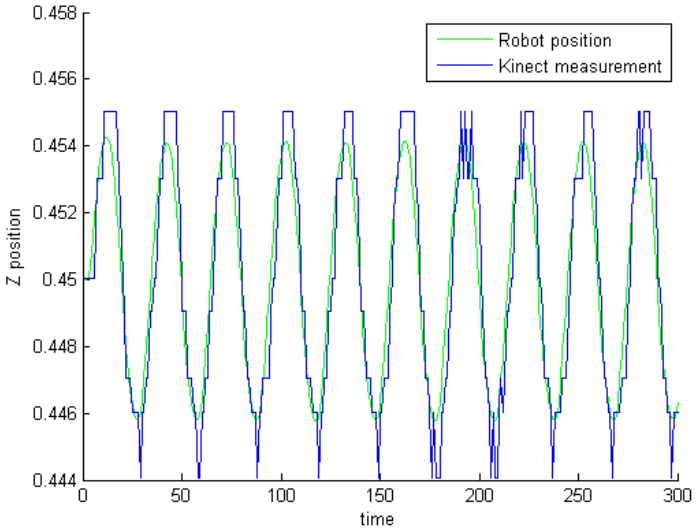


Figure E.1: Kinect tracking with high gain